Oracle® Coherence
User's Guide for Oracle Coherence® Web
Release 3.5
E14536-01

June 2009
# Contents

**Preface** ................................................................................................................................................................................. ix  
Audience ............................................................................................................................................................................................. ix  
Documentation Accessibility ...................................................................................................................................................................... ix  
Related Documents .................................................................................................................................................................................. x  
Conventions ........................................................................................................................................................................................... x  

## 1 Introduction

**What is Coherence*Web?** ......................................................................................................................................................... 1-1  
**Supported Web Containers** ......................................................................................................................................................... 1-2  
**Installation and Deployment Road Map** ...................................................................................................................................... 1-3  
- Choose your Cluster Node Isolation ................................................................................................................................. 1-3  
- Choose your Locking Mode ....................................................................................................................................................... 1-3  
- Choose How to Scope Sessions and Session Attributes .............................................................................................. 1-4  
- Choose When to Clean Up Expired HTTP Sessions ........................................................................................................ 1-4  
- Choose the Installation Method ........................................................................................................................................... 1-4

## 2 Installing Coherence*Web on WebLogic Server 9.2 MP1 and 10.3

**Overview of the Coherence*Web SPI** ........................................................................................................................................ 2-1  
Location of the Coherence*Web SPI ........................................................................................................................................ 2-1  
Requirements for Using the Coherence*Web SPI ...................................................................................................................... 2-1  
Coherence*Web SPI Configurations for the WebLogic Server ........................................................................................................ 2-1  
**Overview Of Configuration and Deployment** .................................................................................................................................. 2-2  
Configuring and Starting a Cache Server .................................................................................................................................. 2-4  
Packaging Applications and Configuring Cluster Nodes ...................................................................................................... 2-4  
- Packaging and Configuring Application Server-Scoped Cluster Nodes .............................................................................. 2-5  
- Packaging and Configuring EAR-Scoped Cluster Nodes ................................................................................................. 2-6  
- Packaging and Configuring WAR-Scoped Cluster Nodes ................................................................................................. 2-6  
Configuring Web Applications for Coherence*Web .................................................................................................................. 2-6  
**Using SAML SSO with Coherence*Web** .................................................................................................................................. 2-9  
**Known Limitations** ........................................................................................................................................................................ 2-10

## 3 Installing Coherence*Web on Other Application Servers

**Installing Coherence*Web Using the WebInstaller** .................................................................................................................. 3-1
Application Server-Specific Installation Instructions ................................................................. 3-1
  Installing on Oracle WebLogic 10.x .......................................................................................... 3-2
  Installing on Caucho Resin 3.1.x ......................................................................................... 3-2
General Instructions for Installing Coherence*Web Session Management Module ............... 3-3
  Coherence*Web WebInstaller Ant Task ................................................................................... 3-5
  Using the WebInstaller Ant task ............................................................................................ 3-5
  Configuring the WebInstaller Ant Task .................................................................................. 3-6
  WebInstaller Ant Task Examples ............................................................................................ 3-6
Testing HTTP Session Management ......................................................................................... 3-7
How the Coherence*Web Installer Instruments a Java EE Application ........................................ 3-8
Installing Coherence*Web into Applications using Java EE Security ......................................... 3-9

4 Coherence*Web Session Management Features

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session Models</td>
<td>4-1</td>
</tr>
<tr>
<td>Traditional Model</td>
<td>4-2</td>
</tr>
<tr>
<td>Monolithic Model</td>
<td>4-3</td>
</tr>
<tr>
<td>Split Model</td>
<td>4-4</td>
</tr>
<tr>
<td>Session Model Recommendations</td>
<td>4-5</td>
</tr>
<tr>
<td>Session and Session Attribute Scoping</td>
<td>4-6</td>
</tr>
<tr>
<td>Session Scoping</td>
<td>4-6</td>
</tr>
<tr>
<td>Preventing Web Applications from Sharing Session Data</td>
<td>4-6</td>
</tr>
<tr>
<td>Keeping Session Cookies Separate</td>
<td>4-7</td>
</tr>
<tr>
<td>Session Attribute Scoping</td>
<td>4-8</td>
</tr>
<tr>
<td>Cluster Node Isolation</td>
<td>4-8</td>
</tr>
<tr>
<td>Application Server-Scoped Cluster Nodes</td>
<td>4-9</td>
</tr>
<tr>
<td>EAR-Scoped Cluster Nodes</td>
<td>4-10</td>
</tr>
<tr>
<td>WAR-Scoped Cluster Nodes</td>
<td>4-11</td>
</tr>
<tr>
<td>Deployment Topologies</td>
<td>4-12</td>
</tr>
<tr>
<td>In-Process</td>
<td>4-14</td>
</tr>
<tr>
<td>Out-of-Process</td>
<td>4-14</td>
</tr>
<tr>
<td>Out-of-Process with Coherence*Extend</td>
<td>4-15</td>
</tr>
<tr>
<td>Managing and Monitoring Applications with JMX</td>
<td>4-16</td>
</tr>
<tr>
<td>Cleaning Up Expired HTTP Sessions</td>
<td>4-19</td>
</tr>
<tr>
<td>Understanding the Session Reaper</td>
<td>4-20</td>
</tr>
<tr>
<td>Configuring the Session Reaper</td>
<td>4-22</td>
</tr>
<tr>
<td>Overriding the Distribution of HTTP Sessions and Attributes</td>
<td>4-22</td>
</tr>
<tr>
<td>Implementing a Session Distribution Controller</td>
<td>4-23</td>
</tr>
<tr>
<td>Registering a Session Distribution Controller Implementation</td>
<td>4-23</td>
</tr>
<tr>
<td>Configuring Coherence<em>Web with Coherence</em>Extend</td>
<td>4-24</td>
</tr>
<tr>
<td>Configuring Coherence*Web for Optimistic Locking</td>
<td>4-24</td>
</tr>
</tbody>
</table>
5 Installing Coherence*Web on WebLogic Portal
   Installing Coherence*Web with WebLogic Portal .......................................................... 5-1

A Coherence*Web Configuration Parameters

B Session Cache Configuration File

Index
## List of Examples

<table>
<thead>
<tr>
<th>Example</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2–1</td>
<td>Library Reference for Each WAR File</td>
<td>2-5</td>
</tr>
<tr>
<td>2–2</td>
<td>Library Reference for Each Web Application in the EAR</td>
<td>2-6</td>
</tr>
<tr>
<td>2–3</td>
<td>Library Reference for the Web Application</td>
<td>2-6</td>
</tr>
<tr>
<td>2–4</td>
<td>Enabling Coherence Web Sessions in web.xml</td>
<td>2-9</td>
</tr>
<tr>
<td>2–5</td>
<td>Ensuring a Unique Session for Each Web Application</td>
<td>2-10</td>
</tr>
<tr>
<td>3–1</td>
<td>Task Import Statement for Coherence*Web WebInstaller</td>
<td>3-5</td>
</tr>
<tr>
<td>4–1</td>
<td>Configuration to Prevent Applications from Sharing Session Data</td>
<td>4-7</td>
</tr>
<tr>
<td>4–2</td>
<td>Sample Session Distribution Controller Implementation</td>
<td>4-23</td>
</tr>
<tr>
<td>4–3</td>
<td>session-cache-config-server.xml File</td>
<td>4-25</td>
</tr>
<tr>
<td>4–4</td>
<td>session-cache-config-client.xml File</td>
<td>4-28</td>
</tr>
<tr>
<td>B–1</td>
<td>Contents of the session-cache-config.xml File</td>
<td>B-2</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2–1</td>
<td>WebLogic Smart Update Login Dialog Box</td>
<td>2-3</td>
</tr>
<tr>
<td>2–2</td>
<td>WebLogic Smart Update Tree Browser</td>
<td>2-3</td>
</tr>
<tr>
<td>4–1</td>
<td>Traditional, Monolithic, and Split Session Models</td>
<td>4-2</td>
</tr>
<tr>
<td>4–2</td>
<td>Traditional Session Model</td>
<td>4-3</td>
</tr>
<tr>
<td>4–3</td>
<td>Monolithic Session Model</td>
<td>4-4</td>
</tr>
<tr>
<td>4–4</td>
<td>Split Session Model</td>
<td>4-5</td>
</tr>
<tr>
<td>4–5</td>
<td>Application Server-Scoped Cluster</td>
<td>4-9</td>
</tr>
<tr>
<td>4–6</td>
<td>EAR-Scoped Cluster</td>
<td>4-10</td>
</tr>
<tr>
<td>4–7</td>
<td>WAR-Scoped Clusters</td>
<td>4-11</td>
</tr>
<tr>
<td>4–8</td>
<td>In-Process Deployment Topology</td>
<td>4-14</td>
</tr>
<tr>
<td>4–9</td>
<td>Out of Process Deployment Topology</td>
<td>4-15</td>
</tr>
<tr>
<td>4–10</td>
<td>Out-of-Process with Coherence*Extend Deployment Topology</td>
<td>4-16</td>
</tr>
<tr>
<td>4–11</td>
<td>HttpSessionManagerMBean Displayed in the JConsole Browser</td>
<td>4-19</td>
</tr>
<tr>
<td>Table</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>1–1</td>
<td>Web Containers Supported by Coherence*Web</td>
<td>1-2</td>
</tr>
<tr>
<td>2–1</td>
<td>Required WebLogic Server and Coherence Patch Release Levels</td>
<td>2-2</td>
</tr>
<tr>
<td>2–2</td>
<td>Parameters that can be Configured in web.xml</td>
<td>2-7</td>
</tr>
<tr>
<td>2–3</td>
<td>HTTP Session Cookie Parameters</td>
<td>2-7</td>
</tr>
<tr>
<td>2–4</td>
<td>Unsupported Coherence*Web Configuration Parameters for WebLogic Server SPI</td>
<td>2-8</td>
</tr>
<tr>
<td>3–1</td>
<td>Settings to Cluster ServletContext Attributes</td>
<td>3-4</td>
</tr>
<tr>
<td>3–2</td>
<td>Settings to Enumerate All Sessions in the Application</td>
<td>3-4</td>
</tr>
<tr>
<td>3–3</td>
<td>Settings to Increase Length of HttpSession ID</td>
<td>3-4</td>
</tr>
<tr>
<td>3–4</td>
<td>Settings to Support URI Encoding</td>
<td>3-4</td>
</tr>
<tr>
<td>3–5</td>
<td>Coherence*Web WebInstaller Ant Task Attributes</td>
<td>3-6</td>
</tr>
<tr>
<td>3–6</td>
<td>Load Balancer Command Line Options</td>
<td>3-8</td>
</tr>
<tr>
<td>4–1</td>
<td>Object Name for the HttpSessionManagerMBean</td>
<td>4-16</td>
</tr>
<tr>
<td>4–2</td>
<td>Information Returned by the HttpSessionManagerMBean</td>
<td>4-17</td>
</tr>
<tr>
<td>4–3</td>
<td>Coherence*Web Parameter Settings for Optimistic Locking</td>
<td>4-24</td>
</tr>
<tr>
<td>4–4</td>
<td>System Property Values for Proxy JVMs</td>
<td>4-25</td>
</tr>
<tr>
<td>4–5</td>
<td>System Property Values for Storage JVMs</td>
<td>4-25</td>
</tr>
<tr>
<td>A–1</td>
<td>Configuration Parameters for Coherence*Web</td>
<td>A-1</td>
</tr>
<tr>
<td>B–1</td>
<td>Cache-Related Values used in session-cache-config.xml</td>
<td>B-1</td>
</tr>
<tr>
<td>B–2</td>
<td>Services-Related Values used in session-cache-config.xml</td>
<td>B-2</td>
</tr>
</tbody>
</table>
Preface

This document describes how to install and configure Coherence®Web.

Audience

This document is intended for application developers who want to be able to manage session state in clustered environments.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Deaf/Hard of Hearing Access to Oracle Support Services

To reach Oracle Support Services, use a telecommunications relay service (TRS) to call Oracle Support at 1.800.223.1711. An Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process. Information about TRS is available at http://www.fcc.gov/cgb/consumerfacts/trs.html, and a list of phone numbers is available at http://www.fcc.gov/cgb/dro/trsphonebk.html.
Related Documents

For more information, see the following documents in the Oracle Coherence documentation set:

- *Getting Started for Oracle Coherence*
- *Developer’s Guide for Oracle Coherence*
- *Client Guide for Oracle Coherence*
- *Tutorial for Oracle Coherence*
- *Integration Guide for Oracle Coherence*

Conventions

The following text conventions are used in this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface</strong></td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td><strong>italic</strong></td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
</tbody>
</table>
This chapter provides general information about Coherence*Web. It describes what Coherence*Web is, what containers it is supported on, and provides a road map for the deployment decisions you should consider based on your environment.

**What is Coherence*Web?**

Coherence*Web is an HTTP session management module dedicated to managing session state in clustered environments. Built on top of Oracle Coherence, Coherence*Web:

- enables session sharing and management across different Web applications, domains and heterogeneous application servers.
- brings Coherence data grid’s data scalability, availability, reliability, and performance to in-memory session management and storage.
- supports all of the mainstream application servers such as Oracle WebLogic Server, IBM WebSphere, Tomcat, and so on (see "Supported Web Containers" on page 1-2).
- supports numerous portal containers, including Oracle WebLogic Portal (see Chapter 5, "Installing Coherence*Web on WebLogic Portal").
- allows for session state to be managed in the various caching topologies available in Coherence (that is, Replicated, Partitioned, Near Caching, Read-Through, Write-Through, Write-Behind and Refresh-Ahead Caching, and so on).
- allows storage of session data outside of the Java EE application server, freeing application server heap space and enabling server restarts without session data loss (see "Deployment Topologies" on page 4-14).
- supports multiple advanced session models (that is, Monolithic, Traditional, and Split Session) which define how the session state is physically managed and serialized/deserialized in the cluster (see “Session Models” on page 4-1).
- Provides advanced session distribution and lifecycle controls by way of the session distribution controller and reaper (see "Cleaning Up Expired HTTP Sessions" on page 4-19).
- supports fine-grained session and session attribute scoping by way of pluggable policies (see "Session and Session Attribute Scoping" on page 4-6).

**Using Coherence*Web with WebLogic Server and WebLogic Portal**

The current release of Coherence*Web integrates with WebLogic Server and WebLogic Portal versions 9.2 MP1 and 10.3 using the native WebLogic session management SPI.
The result of this tighter integration with WebLogic is simplified installation and deployment that no longer requires application instrumentation (using the WebInstaller).

- Chapter 2, "Installing Coherence*Web on WebLogic Server 9.2 MP1 and 10.3," describes the SPI-based WebLogic Server implementation of Coherence*Web in more detail.

- Chapter 5, "Installing Coherence*Web on WebLogic Portal," describes the SPI-based WebLogic Portal implementation of Coherence*Web in more detail.

**Coherence*Web and other Application Servers**

For WebLogic Server versions 10.3 and 9.2MP1, as well as other third-party application servers, Coherence*Web provides a generic installer that transparently instruments your Web applications.

Chapter 3, "Installing Coherence*Web on Other Application Servers," describes the implementation of Coherence*Web with WebInstaller in more detail.

**Supported Web Containers**

Table 1–1 summarizes the Web containers supported by the Coherence*Web Session Management Module. It also provides links to the information required to install Coherence*Web on them. Notice that all of the Web containers (except Oracle WebLogic Server 9.2 MP1 and 10.3) share the same general installation instructions. A few, such as Oracle OC4J, Caucho, and WebLogic 10.x, require extra, container-specific steps that you must complete before starting the general installation instructions.

To install the Coherence*Web Session Management Module on WebLogic Server 9.2 MP1 and 10.3 and later, you can use only SPI-based installation. For instructions on installing the Management Module on WebLogic Server 9.2 MP1 and 10.3, see Chapter 2, "Installing Coherence*Web on WebLogic Server 9.2 MP1 and 10.3."

**Note:** The value in the **Server Type Alias** column is used only by the Coherence*Web WebInstaller installation. The value is passed to the WebInstaller through the `-server` command line option.

<table>
<thead>
<tr>
<th>Application Server</th>
<th>Server Type Alias</th>
<th>See this Installation Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Tomcat 5.5.x</td>
<td>Generic</td>
<td>&quot;General Instructions for Installing Coherence*Web Session Management Module&quot;</td>
</tr>
<tr>
<td>Apache Tomcat 6.0.x</td>
<td>Generic</td>
<td>&quot;General Instructions for Installing Coherence*Web Session Management Module&quot;</td>
</tr>
<tr>
<td>Caucho Resin 3.1.x</td>
<td>Resin/3.1.x</td>
<td>&quot;Installing on Caucho Resin 3.1.x&quot;</td>
</tr>
<tr>
<td>IBM WebSphere 6.x</td>
<td>WebSphere/6.x</td>
<td>&quot;General Instructions for Installing Coherence*Web Session Management Module&quot;</td>
</tr>
<tr>
<td>JBoss Application Server</td>
<td>Generic or Jetty/5.1.x</td>
<td>&quot;General Instructions for Installing Coherence*Web Session Management Module&quot;</td>
</tr>
<tr>
<td>Jetty 5.1.x</td>
<td>Jetty/5.1.x</td>
<td>&quot;General Instructions for Installing Coherence*Web Session Management Module&quot;</td>
</tr>
<tr>
<td>Jetty 6.1.x</td>
<td>Generic</td>
<td>&quot;General Instructions for Installing Coherence*Web Session Management Module&quot;</td>
</tr>
</tbody>
</table>
This section provides a general outline of the deployment decisions you should make before you configure and install Coherence*Web. Coherence*Web is supported on many different application servers. The type of application server that you are deploying on determines whether you will install Coherence*Web using the WebLogic SPI installation or the WebInstaller. Regardless of which application server you are using, you might have to change some of the Coherence*Web configuration options to meet your particular requirements, such as packaging considerations, session model, session locking mode and deployment topology.

### Choose your Cluster Node Isolation

Cluster node isolation refers to the number of nodes that Coherence creates within each application server JVM and where the Coherence library is deployed. A few different isolation modes are supported.

For example: you may be deploying multiple applications to the container that require the use of the same cluster (or one coherence node); you may have multiple Web applications packaged in a single EAR file that want to use a single cluster; or you may have Web applications that must keep their session data separate and must be deployed to their own individual Coherence cluster. These choices and the deployment descriptors and elements that must be configured are described in "Cluster Node Isolation" on page 4-8.

### Choose your Locking Mode

Locking mode refers to the behavior of HTTP sessions when they are accessed concurrently by multiple Web container threads. Coherence*Web offers a number of difference session locking options. For example, you can choose to allow multiple nodes in a cluster to access an HTTP session simultaneously, to not allow more than one node in the cluster to access an HTTP session, or to not allow more than one thread in the cluster to access an HTTP session. You can also allow multiple threads to access the same Web application instance while prohibiting concurrent access by threads in different Web application instances. These choices, and the deployment descriptors and elements that must be configured are described in "Session Locking Modes" on page 4-12.
Choose How to Scope Sessions and Session Attributes

Session and session attribute scoping refers to the fine-grained control over how both session data and session attributes are scoped (or "shared") across application boundaries. Coherence*Web supports sharing sessions across Web applications and restricts which session attributes are shared across the application boundaries. These choices, and the deployment descriptors and elements that must be configured are described in "Session and Session Attribute Scoping" on page 4-6.

Choose When to Clean Up Expired HTTP Sessions

HTTP sessions are eventually cleaned up by the session reaper, and the associated memory is freed. The Coherence*Web session reaper provides a service similar to the JVM's own Garbage Collection (GC) capability: it cleans up HTTP sessions and frees memory once the session has expired. The session reaper is described in "Cleaning Up Expired HTTP Sessions" on page 4-19.

Choose the Installation Method

The installation procedure that you follow depends on your application server. "Supported Web Containers" on page 1-2 provides a list of the application servers supported by Coherence*Web.

- For WebLogic Server 9.2 MP1 and 10.3 and WebLogic Portal 10.3, use the native WebLogic Server SPI-based installation procedure. See Chapter 2, "Installing Coherence*Web on WebLogic Server 9.2 MP1 and 10.3."

  Note that the installation of Coherence*Web on WebLogic Portal 10.3 is completely independent of WebLogic Server; that is, you do not have to install Coherence*Web on WebLogic Server to install it on WebLogic Portal. See Chapter 5, "Installing Coherence*Web on WebLogic Portal."

- For other application servers, use the generic Java EE Web application instrumentation. See Chapter 3, "Installing Coherence*Web on Other Application Servers."
Installing Coherence*Web on WebLogic Server 9.2 MP1 and 10.3

The current release of Coherence*Web provides a deployment option on the WebLogic Server platform that enables a tighter integration with WebLogic Server. The installation and configuration options described in this chapter apply only to WebLogic Server 9.2 MP1 and 10.3 deployments.

Overview of the Coherence*Web SPI

Coherence*Web is not a replacement for WebLogic Server’s in-memory HTTP state replication services. However, Coherence*Web should be considered when an application has large HTTP session state objects, when running into memory constraints due to storing HTTP session object data, or if you have an existing Coherence cluster and want to off-load HTTP Session storage to a Coherence cluster.

The most significant change introduced by this new deployment option is that applications deployed using the Coherence*Web SPI module no longer require an application to be instrumented by the Coherence*Web WebInstaller.

Location of the Coherence*Web SPI

The WebLogic Server Coherence*Web SPI consists of the coherence-web-spi.war file, located in the coherence\lib directory in the Coherence distribution. The coherence.jar file, located in the same directory, is also necessary for enabling Coherence*Web functionality in WebLogic Server.

Requirements for Using the Coherence*Web SPI

The Coherence*Web SPI for WebLogic Server requires that a load balancer which enforces HTTP session JVM affinity is running in front of the WebLogic Server tier. WebLogic Server ships with several different proxy plug-ins which enforce JVM session stickiness. Documentation for configuring the WebLogic Server proxy plug-in is available here:

http://download.oracle.com/docs/cd/E12840_01/wls/docs103/cluster/load_balancing.html#wp1026940

Coherence*Web SPI Configurations for the WebLogic Server

There are two differences between the default cache configuration for the Coherence*Web SPI for WebLogic Server and Coherence*Web:
The Coherence*Web SPI for WebLogic Server is configured with local-storage disabled. This means a Coherence cache server must be running in its own JVM, separate from the JVM running WebLogic Server.

The timeout for requests to the cache server to respond is 30 seconds. If a request to the cache server has not responded in 30 seconds, a com.tangosol.net.RequestTimeoutException exception is thrown.

The Coherence caches used by the Coherence*Web SPI are configured by the session-cache-config.xml file. This file is located inside the coherence-web-spi.war file under the WEB-INF\classes directory. Any cache configuration change should be put inside session-cache-config.xml, and then repackaged inside coherence-web-spi.war.

Overview Of Configuration and Deployment

The Coherence*Web distribution includes a deployable shared library that contains a native plug-in to WebLogic Server's HTTP Session Management interface. To enable Coherence*Web on WebLogic Server for a Web application, complete the following steps:

1. Apply the WebLogic Server publicly available patch to all WebLogic Server instances that are hosting the Web applications that will use Coherence*Web. Table 2–1 lists the appropriate patches for your version of WebLogic Server and Coherence release level.

| Table 2–1 Required WebLogic Server and Coherence Patch Release Levels |
|--------------------------------------------------|-----------------|-----------------|
| WebLogic Smart Update                           | WebLogic Server 9.2 MP1 | WebLogic Server 10.3 |
| Minimum Coherence Release Level/MetaLink Patch ID | 3.4.2 Patch2 - Patch ID: 8429415 | 3.4.2 Patch6 - Patch ID: 11399293 |

The patches can be downloaded by using either the MetaLink Web site or the WebLogic Server’s Smart Update utility.

To Download from MetaLink:

a. Go to the Metalink Web site to manually locate the patch.
   
   http://metalink.oracle.com/

b. Select the Patches tab and click the Simple Search link. On the subsequent screen, submit a search for a Patch Number/Name with the appropriate value (for example, 11399293).

c. Download the patch zip file from the displayed results.

d. See the README.txt included in the patch zip file for instructions for applying the Coherence patch.

To Download with Smart Update:

a. Review the instructions for using Smart Update to install WebLogic Server patches.
   
   http://download.oracle.com/docs/cd/E11035_01/smartUpdate31/guide/install.html#wp1091614

For production environments it is recommended that you review the Smart Update production installation:
Installing Coherence®Web on WebLogic Server 9.2 MP1 and 10.3

http://download.oracle.com/docs/cd/E12840_01/common/smartupdate/guide/remote.html#wp1071859

b. Select Start> All Programs> Oracle WebLogic> Smart Update to open the log in dialog box. Use your Support ID and Password to log in.

Figure 2–1  WebLogic Smart Update Login Dialog Box

![WebLogic Smart Update Login Dialog Box](image)

b. Select Start> All Programs> Oracle WebLogic> Smart Update to open the log in dialog box. Use your Support ID and Password to log in.

Figure 2–1  WebLogic Smart Update Login Dialog Box

![WebLogic Smart Update Login Dialog Box](image)

c. Download and apply the appropriate patch for your version of the WebLogic Server: 6W2W for WLS 10.3, AJQB for WLS 9.2 MP1). Restart the WebLogic Server. Figure 2–2 illustrates the 6W2W patch selected in the Smart Update browser.

Figure 2–2  WebLogic Smart Update Tree Browser

![WebLogic Smart Update Tree Browser](image)
2. (Optional) modify the session-cache-config.xml file to customize the Cache topology for Coherence*Web.

This configuration file is located in the /WEB-INF/classes directory within the coherence-web-spi.war file. If you modify this file, then it should be updated in coherence-web-spi.war.


3. Start a Cache Server Tier in a separate JVM from the one running WebLogic Server.

See "Configuring and Starting a Cache Server" on page 2-4 for more information.

4. Determine the appropriate packaging based on your deployment requirements and follow the packaging instructions.

See "Packaging Applications and Configuring Cluster Nodes" on page 2-4 for more information.

5. (Optional) Modify the web.xml and weblogic.xml files in the WAR deployment if advanced configuration is required for a Web application using Coherence*Web.

Coherence Web parameters that can be configured for Web applications running on the WebLogic Server are described in "Configuring Web Applications for Coherence*Web" on page 2-6. The entire set of Coherence*Web parameters are described in Appendix A, "Coherence*Web Configuration Parameters."

---

**Note:** If you are deploying Coherence*Web in a WebLogic Portal environment, see Chapter 5, "Installing Coherence*Web on WebLogic Portal" for installation instructions.

---

### Configuring and Starting a Cache Server

A Cache Server JVM is a dedicated Coherence JVM that is responsible for storing and managing all cached data (in this case, HttpSession state). One or more cache server JVMs must be started before the WLS/WLP JVMs can be started.

1. Create a script for starting a cache server JVM. The following is an very simple example of a script that starts a storage-enabled cache server for use with Coherence*Web. This example assumes that you are using a Sun JVM. See JVM Tuning in the Developer's Guide for Oracle Coherence for more information. (The following command should be entered on one line but is wrapped here for ease of reading.)

   ```
   java -server -Xms512m -Xmx512m -cp <Coherence installation dir>/lib/coherence.jar:<Coherence installation dir>/lib/coherence-web-spi.war -Dtangosol.coherence.management.remote=true -Dtangosol.coherence.cacheconfig=WEB-INF/classes/session-cache-config.xml -Dtangosol.coherence.session.localstorage=true com.tangosol.net.DefaultCacheServer
   ```

2. Start one or more cache server JVMs using the script described in the previous step.

---

### Packaging Applications and Configuring Cluster Nodes

Coherence cluster nodes are class loader scoped. Therefore, you must configure the number of unique Coherence cluster nodes in a Coherence*Web deployment before
packaging the application(s). The packing and configuration options are described in the following sections:

- Packaging and Configuring Application Server-Scoped Cluster Nodes
- Packaging and Configuring EAR-Scoped Cluster Nodes
- Packaging and Configuring WAR-Scoped Cluster Nodes

You can find detailed information about each of the options under "Cluster Node Isolation" on page 4-8.

You will find the `coherence.jar` and `coherence-web-spi.war` files located in the `/lib` directory of the Coherence distribution.

---

**Note:** The application server-scoped cluster configuration should be considered very carefully and *never* used in environments where application interaction is unknown or unpredictable.

An example of such an environment may be a deployment where multiple application teams are deploying applications written independently, without carefully coordinating and enforcing their conventions and naming standards. With this configuration, all applications are part of the same cluster and the likelihood of collisions between namespaces for caches, services, and other configuration settings is quite high and may lead to unexpected results.

For these reasons, Oracle Coherence strongly recommends that you use EAR-scoped and WAR-scoped cluster node configurations. If you are in doubt regarding which deployment topology to choose, or if this warning applies to your deployment, then *do not* choose the application server-scoped cluster node configuration.

---

**Packaging and Configuring Application Server-Scoped Cluster Nodes**

1. Deploy `coherence-web-spi.war` as a shared library on each WebLogic Server.
2. Edit your WebLogic Server system classpath to include `coherence.jar` or copy the JAR to your `$DOMAIN_HOME/lib` directory.
3. Enable Coherence*Web in your Web application.

   Add the library reference stanza illustrated in Example 2–1 to the `weblogic.xml` in each WAR file deployed in the WebLogic server that intends to use Coherence*Web.

**Example 2–1 Library Reference for Each WAR File**

```xml
<weblogic-web-app>
  ...
  <library-ref>
    <library-name>coherence-web-spi</library-name>
    <specification-version>1.0.0.0</specification-version>
    <implementation-version>1.0.0.0</implementation-version>
    <exact-match>false</exact-match>
  </library-ref>
  ...
</weblogic-web-app>
```
Packaging and Configuring EAR-Scoped Cluster Nodes
1. Deploy coherence-web-spi.war as a shared library on each WebLogic Server.
2. Place coherence.jar in the EAR’s APP-INF/lib directory.

Create a shared library reference in each Web application in the EAR by adding the stanza illustrated in Example 2–2 to the weblogic.xml file:

Example 2–2  Library Reference for Each Web Application in the EAR
<weblogic-web-app>
  ...
  <library-ref>
    <library-name>coherence-web-spi</library-name>
    <specification-version>1.0.0.0</specification-version>
    <implementation-version>1.0.0.0</implementation-version>
    <exact-match>false</exact-match>
  </library-ref>
  ...
</weblogic-web-app>

Packaging and Configuring WAR-Scoped Cluster Nodes
1. Deploy coherence-web-spi.war as a shared library on each WebLogic Server.
2. Place coherence.jar in the WAR’s WEB-INF/lib directory.

Create a shared library reference by adding the stanza illustrated in Example 2–3 to the weblogic.xml file in the Web application’s WEB-INF directory.

Example 2–3  Library Reference for the Web Application
<weblogic-web-app>
  ...
  <library-ref>
    <library-name>coherence-web-spi</library-name>
    <specification-version>1.0.0.0</specification-version>
    <implementation-version>1.0.0.0</implementation-version>
    <exact-match>false</exact-match>
  </library-ref>
  ...
</weblogic-web-app>

Configuring Web Applications for Coherence*Web
Since Coherence*Web is in control of the HTTP session lifecycle, most data from the <session-descriptor> element in either weblogic.xml or weblogic-application.xml is ignored.

The Coherence*Web SPI ships with a configuration that should be sufficient for most Web applications. If you must make any changes to the configuration or override any previous settings, you can apply any of the Coherence*Web parameters described in Appendix A, "Coherence*Web Configuration Parameters." You can apply these parameters by using the <context-param> element in the web.xml file.

Table 2–2 lists only those Coherence*Web parameters which have a WebLogic-specific default. For full descriptions of all Coherence*Web parameters, see Appendix A, "Coherence*Web Configuration Parameters."
Table 2–2  Parameters that can be Configured in web.xml

<table>
<thead>
<tr>
<th>Parameter</th>
<th>WebLogic-Specific Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>coherence-sessioncollection-class</td>
<td>If unspecified, defaults to com.tangosol.coherence.servlet.SplitHttpSessionCollection.</td>
</tr>
<tr>
<td>coherence-reaperdaemon-assume-localeity</td>
<td>If unspecified, defaults to false</td>
</tr>
<tr>
<td>coherence-session-member-locking</td>
<td>If unspecified, defaults to true</td>
</tr>
<tr>
<td>coherence-session-app-locking</td>
<td>If unspecified, defaults to true.</td>
</tr>
<tr>
<td>coherence-sticky-sessions</td>
<td>If unspecified, defaults to true.</td>
</tr>
<tr>
<td>coherence-scopecontroller-class</td>
<td>If unspecified, defaults to com.tangosol.coherence.servlet.AbstractHttpSessionCollection$ApplicationScopeController</td>
</tr>
<tr>
<td>coherence-preserve-attributes</td>
<td>If unspecified, defaults to true</td>
</tr>
</tbody>
</table>

Table 2–3 describes the generated HTTP session cookie parameters that can be configured in weblogic.xml or weblogic-application.xml file using the <session-descriptor> element.

Table 2–3  HTTP Session Cookie Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cookie-comment</td>
<td>null</td>
<td>Specifies the comment that identifies the session tracking cookie in the cookie file.</td>
</tr>
<tr>
<td>cookie-domain</td>
<td>null</td>
<td>Specifies the domain for which the cookie is valid. For example, setting cookie-domain to .mydomain.com returns cookies to any server in the *.mydomain.com domain. The domain name must have at least two components. Setting a name to *.com or *.net is not valid. If not set, this attribute defaults to the server that issued the cookie. For more information, see Cookie.setDomain() in the Servlet specification from Sun Microsystems.</td>
</tr>
<tr>
<td>cookies-enabled</td>
<td>true</td>
<td>Use of session cookies is enabled by default and is recommended, but you can disable them by setting this property to false. You might turn this option off to test.</td>
</tr>
<tr>
<td>cookie-max-age-secs</td>
<td>-1</td>
<td>Sets the life span of the session cookie, in seconds, after which it expires on the client. The default value is -1 (unlimited). For more information about cookies, see Using Sessions and Session Persistence.</td>
</tr>
</tbody>
</table>
In the WebLogic SPI module, the Coherence*Web configuration parameters listed in Table 2–4 are not controlled by Coherence*Web and must be specified as outlined in the table.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>coherence-servletcontext-clustered</td>
<td>Not Supported.</td>
<td></td>
</tr>
<tr>
<td>coherence-servletcontext-cachename</td>
<td>Not Supported.</td>
<td></td>
</tr>
<tr>
<td>coherence-eventlisteners</td>
<td>Not Supported.</td>
<td></td>
</tr>
<tr>
<td>coherence-enable-sessioncontext</td>
<td>Not Supported.</td>
<td></td>
</tr>
<tr>
<td>coherence-session-cookies-enabled</td>
<td>This value is set by the WebLogic session-descriptor cookies-enabled element in weblogic.xml or weblogic-application.xml.</td>
<td></td>
</tr>
<tr>
<td>coherence-session-cookie-domain</td>
<td>This value is set by the WebLogic session-descriptor cookie-domain element in weblogic.xml or weblogic-application.xml.</td>
<td></td>
</tr>
<tr>
<td>coherence-session-cookie-path</td>
<td>This value is set by the WebLogic session-descriptor cookie-path element in weblogic.xml or weblogic-application.xml.</td>
<td></td>
</tr>
<tr>
<td>coherence-session-cookie-max-age</td>
<td>This value is set by the WebLogic session-descriptor cookie-max-age-secs element in weblogic.xml or weblogic-application.xml.</td>
<td></td>
</tr>
<tr>
<td>coherence-urlencode-enabled</td>
<td>Not Supported.</td>
<td></td>
</tr>
</tbody>
</table>

Table 2–3 (Cont.) HTTP Session Cookie Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cookie-path</td>
<td>null</td>
<td>Defines the session tracking cookie path. If not set, this attribute defaults to / (slash), where the browser sends cookies to all URLs served by WebLogic Server. You may set the path to a narrower mapping, to limit the request URLs to which the browser sends cookies.</td>
</tr>
<tr>
<td>cookie-secure</td>
<td>false</td>
<td>Tells the browser to only send the cookie back over an HTTPS connection. This ensures that the cookie ID is secure and should only be used on Web sites that use HTTPS. Session Cookies over HTTP no longer work if this feature is enabled. You should disable the url-rewriting-enabled element if you intend to use this feature.</td>
</tr>
<tr>
<td>id-length</td>
<td>52</td>
<td>Sets the size of the session ID. The minimum value is 8 bytes and the maximum value is Integer.MAX_VALUE. If you are writing a WAP application, you must use URL rewriting because the WAP protocol does not support cookies. Also, some WAP devices have a 128-character limit on URL length (including attributes), which limits the amount of data that can be transmitted using URL rewriting. To allow more space for attributes, use this attribute to limit the size of the session ID that is randomly generated by WebLogic Server. You can also limit the length to a fixed 52 characters, and disallow special characters, by setting the WAPEnabled attribute. For more information, see URL Rewriting and Wireless Access Protocol in Developing Web Applications for WebLogic Server.</td>
</tr>
</tbody>
</table>
Using SAML SSO with Coherence*Web

WebLogic Server provides basic single sign-on (SSO) functionality by default. To use SAML SSO functionality with Coherence*Web, you must modify the contents of the saml2.war Web application.

1. Backup the existing saml2.war in the WebLogic Server installation.
2. Un-jar the saml2.war into a temporary directory.
   
   The saml2.war is located in the $WL_SERVER_HOME/server/lib directory.

3. Create a lib directory and a classes directory under the WEB-INF directory.
4. Un-jar the coherence-web-spi.war to retrieve coherence-web.jar and coherence-web-spi.jar. Copy these two jars to the /WEB-INF/lib directory.
5. Copy the session-cache-config.xml file, located in the /WEB-INF/classes directory from the un-jarred coherence-web-spi.war, into the /WEB-INF/classes directory.
6. Place the coherence.jar in the appropriate location, based on the cluster node scoping you selected: application server-, EAR-, or WAR-scoped.
7. Add the code in Example 2–4 to the /WEB-INF/web.xml file:

   Example 2–4  Enabling Coherence Web Sessions in web.xml

   ```xml
   <?xml version="1.0" encoding="UTF-8"?>
   <web-app>
   ...
   <context-param>
     <param-name>coherence-web-sessions-enabled</param-name>
     <param-value>true</param-value>
   </context-param>
   ...
   </web-app>
   ```
8. Re-assemble the saml2.war by using the jar command, for example:
   
   `jar cvf saml2.war $tempdir`

9. Backup the existing saml2.war in the WebLogic Server installation.
10. Replace the saml2.war in the WebLogic installation with the modified saml2.war file.

### Table 2–4 (Cont.) Unsupported Coherence*Web Configuration Parameters for WebLogic Server SPI

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Alternative configuration setting to be used</th>
</tr>
</thead>
<tbody>
<tr>
<td>coherence-session-urlencode-name</td>
<td>Not Supported.</td>
</tr>
<tr>
<td>coherence-session-urldecode-bycontainer</td>
<td>Not Supported.</td>
</tr>
<tr>
<td>coherence-session-urlencode-bycontainer</td>
<td>Not Supported.</td>
</tr>
<tr>
<td>coherence-session-id-length</td>
<td>This value is set by the WebLogic session-descriptor id-length element in weblogic.xml or weblogic-application.xml.</td>
</tr>
</tbody>
</table>

Using SAML SSO with Coherence*Web

WebLogic Server provides basic single sign-on (SSO) functionality by default. To use SAML SSO functionality with Coherence*Web, you must modify the contents of the saml2.war Web application.

1. Backup the existing saml2.war in the WebLogic Server installation.
2. Un-jar the saml2.war into a temporary directory.
   
   The saml2.war is located in the $WL_SERVER_HOME/server/lib directory.

3. Create a lib directory and a classes directory under the WEB-INF directory.
4. Un-jar the coherence-web-spi.war to retrieve coherence-web.jar and coherence-web-spi.jar. Copy these two jars to the /WEB-INF/lib directory.
5. Copy the session-cache-config.xml file, located in the /WEB-INF/classes directory from the un-jarred coherence-web-spi.war, into the /WEB-INF/classes directory.
6. Place the coherence.jar in the appropriate location, based on the cluster node scoping you selected: application server-, EAR-, or WAR-scoped.
7. Add the code in Example 2–4 to the /WEB-INF/web.xml file:

   Example 2–4  Enabling Coherence Web Sessions in web.xml

   ```xml
   <?xml version="1.0" encoding="UTF-8"?>
   <web-app>
   ...
   <context-param>
     <param-name>coherence-web-sessions-enabled</param-name>
     <param-value>true</param-value>
   </context-param>
   ...
   </web-app>
   ```
8. Re-assemble the saml2.war by using the jar command, for example:
   
   `jar cvf saml2.war $tempdir`

9. Backup the existing saml2.war in the WebLogic Server installation.
10. Replace the saml2.war in the WebLogic installation with the modified saml2.war file.
Known Limitations

By default, Coherence*Web creates a single HTTP Session across all Web applications for each client and scopes the session attributes to each Web application. This means that if a session is invalidated in one Web application, that same session is invalidated for all Web applications in WebLogic Server using Coherence*Web.

This functionality requires that the session cookie path is set to "/", making the same session cookie available to all Web applications. If you do not want this behavior, a potential work-around is to reduce the scope the session cookie by adding the entry illustrated in Example 2–5 to the weblogic.xml file in each Web application.

Example 2–5 Ensuring a Unique Session for Each Web Application

```xml
<weblogic-web-app>
  ...
  <session-descriptor>
    <cookie-path>[/path of web-app context, for example "/mainApp/subApp"]</cookie-path>
  </session-descriptor>
  ...
</weblogic-web-app>
```

This ensures a unique session is created for each Web application. It is also possible to scope the session to all Web applications within an EAR file by setting the session cookie path to the context root of the deployed EAR.

This work-around does not work if you deploy an EAR or Web application with "/" as the context path, or if you require WebLogic SSO. WebLogic SSO requires that the session cookie path be set to "/".

This chapter provides instructions on how to use the Coherence*Web WebInstaller to install Coherence*Web for Java EE applications on a variety of different application servers.

**Before Proceeding:** Consult the "Supported Web Containers" on page 1-2 to see if you must perform any application server-specific installation steps.

When deploying Coherence*Web on WebLogic Server you now have two options:

- Use the WebInstaller approach described in this chapter
- Use the SPI-based installation for WebLogic Server 9.2 MP1, 10.3, or later. See Chapter 2, "Installing Coherence*Web on WebLogic Server 9.2 MP1 and 10.3."

### Installing Coherence*Web Using the WebInstaller

Coherence*Web can be enabled for Java EE applications on a number of different Web containers. To enable Coherence*Web, you must run the ready-to-deploy application through the automated Coherence*Web WebInstaller before deploying it. The automated installer prepares the application for deployment. It performs the installation process in two discrete steps: an inspect step and an install step. For more information on what the installer does during these steps, see "How the Coherence*Web Installer Instruments a Java EE Application" on page 3-8.

The installer can be run either from the Java command line or from Ant tasks. The following sections describe the Java command line method. For Ant task-based installation, see "Coherence*Web WebInstaller Ant Task" on page 3-5.

### Application Server-Specific Installation Instructions

All of the Web containers listed in "Supported Web Containers" on page 1-2 that can be installed with the WebInstaller share the same general installation instructions. These instructions are described in "General Instructions for Installing Coherence*Web Session Management Module" on page 3-3.

A few of the Web containers, such as Oracle OC4J, Caucho, and WebLogic 10.x, require extra, container-specific steps that you must complete before starting the general
installation instructions. The following sections describe application server-specific installation steps.

- Installing on Oracle WebLogic 10.x
- Installing on Caucho Resin 3.1.x

Installing on Oracle WebLogic 10.x

Complete the following steps to install the Coherence*Web Session Management Module into an Oracle WebLogic 10-10.2 server:

1. From within the Coherence library directory, extract the `coherence-web.jar` from the `webInstaller.jar`:
   ```
   jar -xvf webInstaller.jar web-install/coherence-web.jar
   ```
   This command extracts the `coherence-web.jar` file into a subdirectory named `web-install`. Use the following commands to move the `coherence-web.jar` file up one level into the library directory:

   **On Windows:**
   ```
   move web-install\coherence-web.jar .
   rmdir web-install
   ```

   **On UNIX:**
   ```
   mv web-install/coherence-web.jar .
   rmdir web-install
   ```

2. For each WebLogic 10.x installation that will be running in the server cluster, update the libraries using the following command (note that it is broken up into multiple lines here only for formatting purposes; this is a single command entered on one line):
   ```
   java -cp coherence.jar;coherence-web.jar com.tangosol.coherence.servlet.WebPluginInstaller <wls-home-path> -install
   ```
   For example, on Windows:
   ```
   java -cp coherence.jar;coherence-web.jar com.tangosol.coherence.servlet.WebPluginInstaller C:\bea\weblogic\wlserver_10 -install
   ```

3. Follow the instructions described in "General Instructions for Installing Coherence*Web Session Management Module" on page 3-3 to complete the installation. Use the value `WebLogic/10.x` for the `server type`.

Installing on Caucho Resin 3.1.x

Complete the following steps to install the Coherence*Web Session Management Module into a Caucho Resin 3.1.x server:

1. From within the Coherence library directory, extract the `coherence-web.jar` from the `webInstaller.jar`:
   ```
   jar -xvf webInstaller.jar web-install/coherence-web.jar
   ```
   This command extracts the `coherence-web.jar` file into a subdirectory named `web-install`. Use the following commands to move the `coherence-web.jar` file up one level into the library directory:

   **On Windows:**
   ```
   move web-install\coherence-web.jar .
   rmdir web-install
   ```

   **On UNIX:**
   ```
   mv web-install/coherence-web.jar .
   rmdir web-install
   ```
move web-install\coherence-web.jar .
rmdir web-install

On UNIX:

mv web-install/coherence-web.jar .
rmdir web-install

2. For each Resin installation that will be running in the server cluster, update the libraries using the following command (note that it is broken up into multiple lines only for formatting purposes; this is a single command entered on one line):

```
java -cp coherence.jar;coherence-web.jar
com.tangosol.coherence.servlet.WebPluginInstaller <resin-home-path> -install
```

For example, on Windows:

```
java -cp coherence.jar;coherence-web.jar
com.tangosol.coherence.servlet.WebPluginInstaller C:\opt\resin31 -install
```

3. Follow the instructions described in "General Instructions for Installing Coherence*Web Session Management Module" on page 3-3 to complete the installation. Use the value Resin/3.1.x for the server type.

**General Instructions for Installing Coherence*Web Session Management Module**

You must complete the following steps to install Coherence*Web for a Java EE application on any of the Web containers listed under "Supported Web Containers" on page 1-2.

If you are installing Coherence*Web Session Management Module on an Oracle OC4J, Caucho, or WebLogic container, then you must complete certain container-specific installation steps before you start the general installation instructions. These container-specific installation steps are described in "Application Server-Specific Installation Instructions" on page 3-1.

To install Coherence*Web for the Java EE application you are deploying:

1. Make sure that the application directory and the .ear file or .war file are not being used or accessed by another process.

2. Change the current directory to the Coherence library directory (%COHERENCE_HOME%\lib on Windows and $COHERENCE_HOME/lib on UNIX).

3. Make sure that the paths are configured so that Java commands will run.

4. Complete the application inspection step by running the following command. Specify the full path to your application and the name of your server found in Table 1–1 (replacing the <app-path> and <server-type> with them in the command line below):

```
java -jar webInstaller.jar <app-path> -inspect -server:<server-type>
```

The system will create (or update, if it already exists), the coherence-web.xml configuration descriptor file for your Java EE application in the directory where the application is located. This configuration descriptor contains the default Coherence*Web settings for your application recommended by the installer.

5. You may proceed to the install step (Step 6) or review and modify the Coherence*Web settings based on your requirements, before running the install step.
You modify the Coherence*Web settings by editing the `coherence-web.xml` descriptor. Appendix A, "Coherence*Web Configuration Parameters," describes the Coherence*Web settings that can be modified. Use the `param-name` and `param-value` subelements of `context-param` to enable the features you want.

For example:

- The setting in Table 3-1 will cluster all `ServletContext` ("global") attributes so that servers in a cluster share the same values for those attributes, and also receive the events specified by the Servlet Specification when those attributes change:

  **Table 3-1 Settings to Cluster ServletContext Attributes**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>param-name</td>
<td>coherence-servletcontext-clustered</td>
</tr>
<tr>
<td>param-value</td>
<td>true</td>
</tr>
</tbody>
</table>

- The setting in Table 3-2 allows an application to enumerate all of the sessions that exist within the application, or to obtain any one of those sessions to examine or manipulate:

  **Table 3-2 Settings to Enumerate All Sessions in the Application**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>param-name</td>
<td>coherence-enable-sessioncontext</td>
</tr>
<tr>
<td>param-value</td>
<td>true</td>
</tr>
</tbody>
</table>

- The setting in Table 3-3 enables you to increase the length of the `HttpSession` ID, which is generated using a `SecureRandom` algorithm; the length can be any value, although in practice it should be small enough to fit into a cookie or a URL (depending on how session IDs are maintained.) Increasing the length can decrease the chance of a session being purposefully hijacked:

  **Table 3-3 Settings to Increase Length of HttpSession ID**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>param-name</td>
<td>coherence-session-id-length</td>
</tr>
<tr>
<td>param-value</td>
<td>32</td>
</tr>
</tbody>
</table>

- By default, the `HttpSession` ID is managed in a cookie. If the application supports URL encoding, set the option described in Table 3-4 to enable it:

  **Table 3-4 Settings to Support URI Encoding**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>param-name</td>
<td>coherence-session-urlencode-enabled</td>
</tr>
<tr>
<td>param-value</td>
<td>true</td>
</tr>
</tbody>
</table>

After double-checking that these changes have been made, save the file and exit the editor. Remember to return back to the Coherence library directory if you are working from a shell or command line.
6. Perform the Coherence*Web application installation step by running the following command, replacing `<app-path>` with the full path to your application:

```java
java -jar webInstaller.jar <app-path> -install
```

The installer requires a valid `coherence-web.xml` configuration descriptor for its use in the same directory in which the application is located.

7. Deploy the updated application and verify that everything functions as expected, using the load balancer if necessary. Remember that the load balancer is intended only for testing and should not be used in a production environment.

---

**Coherence*Web WebInstaller Ant Task**

The Coherence*Web WebInstaller Ant task enables you to run the installer from within your existing Ant build files.

This section contains the following information:

- Using the WebInstaller Ant task
- Configuring the WebInstaller Ant Task
- WebInstaller Ant Task Examples

**Using the WebInstaller Ant task**

To use the Coherence*Web WebInstaller Ant task, add the task import statement illustrated in Example 3–1 to your Ant build file. In this example, `${coherence.home}` refers to the root directory of your Coherence installation.

**Example 3–1  Task Import Statement for Coherence*Web WebInstaller**

```xml
<taskdef name="cwi" classname="com.tangosol.coherence.misc.CoherenceWebAntTask">
  <classpath>
    <pathelement location="${coherence.home}/lib/webInstaller.jar"/>
  </classpath>
</taskdef>
```

The following procedure describes the basic process of installing Coherence*Web into a Java EE application from an Ant build.

1. Build your Java EE application as you normally would.
2. Run the Coherence*Web Ant task with the `operations` attribute set to `inspect`.
3. Make any necessary changes to the generated Coherence*Web XML descriptor.
4. Run the Coherence*Web Ant task with the `operations` attribute set to `install`.

If you are performing iterative development on your application (such as modifying JSPs, Servlets, static resources, and so on), use the following installation process:

1. Run the Coherence*Web Ant task with the `operations` attribute set to `uninstall`, the `failonerror` attribute set to `false`, and the `descriptor` attribute set to the location of the previously generated Coherence*Web XML descriptor (from Step 2 above).
2. Build your Java EE application as you normally would.
3. Run the Coherence*Web Ant task with the `operations` attribute set to `inspect`, `install` and the `descriptor` attribute set to the location of the previously generated Coherence*Web XML descriptor (from Step 2 above).
To change the Coherence*Web configuration settings of a Java EE application that has Coherence*Web installed, use this procedure:

1. Run the Coherence*Web Ant task with the `operations` attribute set to `uninstall` and the `descriptor` attribute set to the location of the Coherence*Web XML descriptor for the Java EE application.

2. Change the necessary configuration parameters in the Coherence*Web XML descriptor.

3. Run the Coherence*Web Ant task with the `operations` attribute set to `install` and the `descriptor` attribute set to the location of the modified Coherence*Web XML descriptor (from Step 2).

### Configuring the WebInstaller Ant Task

Table 3–5 describes the attributes that can be used with the Coherence*Web WebInstaller Ant Task.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>app</code></td>
<td>Path to the target Java EE application. This can be a path to a WAR file, an EAR file, an expanded WAR directory, or an expanded EAR directory.</td>
<td>Yes, if the <code>operations</code> attribute is set to any value other than <code>version</code>.</td>
</tr>
<tr>
<td><code>backup</code></td>
<td>Path to a directory that holds a backup of the original target Java EE application. This attribute defaults to the directory that contains the Java EE application.</td>
<td>No</td>
</tr>
<tr>
<td><code>descriptor</code></td>
<td>Path to the Coherence*Web XML descriptor. This attribute defaults to <code>coherence-web.xml</code> in the directory that contains the target Java EE application.</td>
<td>No</td>
</tr>
<tr>
<td><code>failonerror</code></td>
<td>Stop the Ant build if the Coherence*Web installer exits with a status other than 0. The default is <code>true</code>.</td>
<td>No</td>
</tr>
<tr>
<td><code>nowarn</code></td>
<td>Suppress warning messages. This attribute can be either <code>true</code> or <code>false</code>. The default is <code>false</code>.</td>
<td>No</td>
</tr>
<tr>
<td><code>operations</code></td>
<td>A comma- or space-separated list of operations to perform; each operation must be one of <code>inspect</code>, <code>install</code>, <code>uninstall</code>, or <code>version</code>.</td>
<td>Yes</td>
</tr>
<tr>
<td><code>server</code></td>
<td>The alias of the target Java EE application server.</td>
<td>No</td>
</tr>
<tr>
<td><code>touch</code></td>
<td>Touch JSPs and TLDs that are modified by the Coherence*Web installer. This attribute can be either <code>true</code>, <code>false</code>, or <code>h/d/y h:mm </code>a<code>The default is</code>false`.</td>
<td>No</td>
</tr>
<tr>
<td><code>verbose</code></td>
<td>Show verbose output. This attribute can be either <code>true</code> or <code>false</code>. The default is <code>false</code>.</td>
<td>No</td>
</tr>
</tbody>
</table>

### WebInstaller Ant Task Examples

- Inspect the `myWebApp.war` Web application and generate a Coherence*Web XML descriptor called `my-coherence-web.xml` in the current working directory:

  ```xml
  <cwi app="myWebApp.war" operations="inspect" descriptor="my-coherence-web.xml"/>
  ```

- Install Coherence*Web into the `myWebApp.war` Web application using the Coherence*Web XML descriptor called `my-coherence-web.xml` found in the current working directory:

  ```xml
  <cwi app="myWebApp.war" operations="install" descriptor="my-coherence-web.xml"/>
  ```
Installing Coherence*Web Using the WebInstaller

Testing HTTP Session Management

Coherence comes with a light-weight software load balancer; it is intended only for testing purposes. The load balancer is very useful when testing functionality such as session management and is very easy to use.

1. Start multiple application server processes on one or more server machines, each running your application on a unique IP address and port combination.

2. Open a command (or shell) window.

3. Change the current directory to the Coherence library directory (%COHERENCE_HOME%\lib on Windows and $COHERENCE_HOME/lib on UNIX).

4. Make sure that paths are configured so that Java commands will run.

5. Start the software load balancer with the following command lines (each of these command lines makes the application available on the default HTTP port, which is port 80).

   For example, to test load-balancing locally on one machine with two application server instances on ports 7001 and 7002:

   ```
   java -jar coherence-loadbalancer.jar localhost:80 localhost:7001 localhost:7002
   ```

   To run the load-balancer locally on a machine named server1 that load balances to port 7001 on server1, server2, and server3:

   ```
   java -jar coherence-loadbalancer.jar server1:80 server1:7001 server2:7001 server3:7001
   ```
Assuming the above command line, an application that previously was accessed with the URL http://server1:7001/my.jsp would now be accessed with the URL http://server1:80/my.jsp or just http://server1/my.jsp.

___

**Note:** Make sure that your application uses only relative re-directs or the address of the load-balancer.

___

Table 3–6 describes the command line options for the load balancer:

**Table 3–6 Load Balancer Command Line Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>backlog</td>
<td>Sets the TCP/IP accept backlog option to the specified value, for example: -backlog=64</td>
</tr>
<tr>
<td>random</td>
<td>Specifies the use of a random load-balancing algorithm (default).</td>
</tr>
<tr>
<td>roundrobin</td>
<td>Specifies the use of a round-robin load-balancing algorithm</td>
</tr>
<tr>
<td>threads</td>
<td>Uses the specified number of request/response thread pairs (so the total number of additional daemon threads will be two times the specified value), for example: -threads=64</td>
</tr>
</tbody>
</table>

---

**How the Coherence*Web Installer Instruments a Java EE Application**

During the inspect step, the Coherence*Web WebInstaller performs the following tasks:

1. Generates a template `coherence-web.xml` configuration file that contains basic information about the application and target Web container along with a set of default Coherence*Web configuration context parameters appropriate for the target Web container. See Appendix A, "Coherence*Web Configuration Parameters" for descriptions of all possible parameters.

   If an existing `coherence-web.xml` configuration file exists (for example, from a previous run of the Coherence*Web Installer), the context parameters in the existing file are merged with those in the generated template.

2. Enumerates the JSPs from each Web application in the target Java EE application and adds information about each JSP to the `coherence-web.xml` configuration file.

3. Enumerates the TLDs from each Web application in the target Java EE application and adds information about each TLD to the `coherence-web.xml` configuration file.

During the install step, the Coherence*Web WebInstaller performs the following tasks:

1. Creates a backup of the original Java EE application so that it can be restored during the uninstall step.

2. Adds the Coherence*Web configuration context parameters generated in Step (1) of the inspect step to the `web.xml` descriptor of each Web application contained in the target Java EE application.

3. Unregisters any application-specific `ServletContextListener`, `ServletContextAttributeListener`, `ServletRequestListener`, `ServletRequestAttributeListener`, `HttpSessionListener`, and `HttpSessionAttributeListener` classes (including those registered by TLDs) from each Web application.
4. Registers a Coherence*Web ServletContextListener in each web.xml descriptor. At run time, the Coherence*Web ServletContextListener propagates each ServletContextEvent to each application-specific ServletContextListener.

5. Registers a Coherence*Web ServletContextAttributeListener in each web.xml descriptor. At run time, the Coherence*Web ServletContextAttributeListener propagates each ServletContextAttributeEvent to each application-specific ServletContextAttributeListener.

6. Wraps each application-specific Servlet declared in each web.xml descriptor with a Coherence*Web SessionServlet. At run time, each Coherence*Web SessionServlet delegates to the wrapped Servlet.

7. Adds the following directive to each JSP enumerated in Step (2) of the inspect step:

```jsp
<%@ page extends="com.tangosol.coherence.servlet.api22.JspServlet" %>
```

During the uninstall step, the Coherence*Web WebInstaller replaces the instrumented Java EE application with the backup of the original version created in Step (1) of the install process.

## Installing Coherence*Web into Applications using Java EE Security

**Note:** This section does not apply to the native WebLogic SPI implementation of Coherence*Web. It applies only if you are using the WebInstaller to install Coherence*Web into an application that uses Java EE security.

If you want to install Coherence*Web into an application that uses Java EE security, you must follow these additional steps during installation:

1. Enable Coherence*Web session cookies.
   
   See the `coherence-session-cookies-enabled` configuration element in Table A–1 for additional details.

2. Change the Coherence*Web session cookie name to a name which is different from the one used by the target Web container.

   By default, most containers use `JSESSIONID` for the session cookie name, so a good choice for the Coherence*Web session cookie name is `CSESSIONID`. See the `coherence-session-cookie-name` configuration element in Table A–1 for additional details.

3. Enable session replication for the target Web container.

   If session replication is not enabled, or the container does not support a form of session replication, then you will be forced to re-authenticate to the Web application during failover. See your Web container's documentation for instructions on enabling session replication.

This configuration causes two sessions to be associated with a given authenticated user:

- A Coherence*Web session which contains all session data created by the Web application.
A session created by the Web container during authentication which only stores information necessary to identify the user.
Coherence*Web can be configured in many ways to meet the demands of your environment. Consequently, you might have to change some default configuration options. The purpose of this chapter is to provide an in-depth look at the features that Coherence*Web supports so that you can make the appropriate configuration and deployment decisions.

- **Session Models**
- **Session and Session Attribute Scoping**
- **Cluster Node Isolation**
- **Session Locking Modes**
- **Deployment Topologies**
- **Managing and Monitoring Applications with JMX**
- **Cleaning Up Expired HTTP Sessions**
- **Overriding the Distribution of HTTP Sessions and Attributes**
- **Configuring Coherence*Web with Coherence*Extend**

### Session Models

A session model describes how Coherence*Web physically represents and stores session state in Coherence. Coherence*Web supports a flexible data management model for session state. The session state is managed by an HttpSessionModel object, and the list of all sessions is managed by an HttpSessionCollection object. Coherence*Web includes these different session model implementations out of the box:

- **Traditional Model**—Stores all session state as a single entity but serializes and deserializes attributes individually.

- **Monolithic Model**—Stores all session state as a single entity, serializing and deserializing all attributes as a single operation.

- **Split Model**—Extends the Traditional Model but separates the larger session attributes into independent physical entities.
Figure 4–1  Traditional, Monolithic, and Split Session Models

Traditional Model

The TraditionalHttpSessionModel and TraditionalHttpSessionCollection manage all of the HTTP session data for a particular session in a single Coherence cache entry, but manage each HTTP session attribute (particularly, its serialization and deserialization) separately.

This model is suggested for applications with relatively small HTTP session objects (10KB or less) that do not have issues with object-sharing between session attributes. (Object-sharing between session attributes occurs when multiple attributes of a session have references to the same exact object, meaning that separate serialization and deserialization of those attributes cause multiple instances of that shared object to exist when the HTTP session is later deserialized.)
**Figure 4–2  Traditional Session Model**

**Monolithic Model**

MonolithicHttpSessionModel and MonolithicHttpSessionCollection are similar to the Traditional Model, except that they solve the shared object issue by serializing and deserializing all attributes into a single object stream.

As a result, the Monolithic Model is often less performant than the Traditional Model.
**Split Model**

`SplitHttpSessionModel` and `SplitHttpSessionCollection` manage the core HTTP session data such as the session ID, creation time, last access time, and so on, with all of the small session attributes in the same manner as the Traditional Model, thus ensuring high performance by keeping that block of session data small. All large attributes are split out into separate cache entries to be managed individually, thus supporting very large HTTP session objects without unduly increasing the amount of data that must be accessed and updated within the cluster on each request. In other words, only the large attributes that are modified within a particular request incur any network overhead for their updates, and (because it uses Near Caching) the Split Model generally does not incur any network overhead for accessing either the core HTTP session data or any of the session attributes.
**Session Model Recommendations**

- The Split Model is the recommended session model for most applications.
- The Traditional Model may be more optimal for applications that are known to have small HTTP session objects.
- The Monolithic Model is designed to solve a specific class of problems related to multiple session attributes that have references to the same shared object, and that must maintain that object as a shared object.

*Session Management for Clustered Applications* in *Getting Started with Oracle Coherence*, provides information on the behavior of these models in a clustered environment.

---

**Note:** For configuration information, see Appendix A, "Coherence*Web Configuration Parameters."
Session and Session Attribute Scoping

Coherence*Web allows fine-grained control over how both session data and session attributes are scoped (or "shared") across application boundaries:

Session Scoping

Coherence*Web allows session data to be shared by different Web applications deployed in the same or different Web containers. To do so, you must correctly configure the Coherence*Web cookie context parameters and make the classes of objects stored in session attributes available to each Web application.

If you are using cookies to store session IDs (that is, you are not using URL rewriting), you must set the `coherence-session-cookie-path` context parameter to a common context path of all Web applications that share session data. For example, to share session data between two Web applications registered under the contexts paths `/web/HRPortal` and `/web/InWeb, you should set the `coherence-session-cookie-path` parameter to `/web`. On the other hand, if the two Web applications are registered under the contexts paths `/HRPortal` and `/InWeb, you should set the `coherence-session-cookie-path` parameter to `/`.

If the Web applications that you would like to share session data are deployed on different Web containers running on different machines (that are not behind a common load balancer), you must also set the `coherence-session-cookie-domain` parameter to a domain shared by the machines. For example, to share session data between two Web applications running on `server1.mydomain.com` and `server2.mydomain.com`, you must set the `coherence-session-cookie-domain` parameter to `.mydomain.com`.

To correctly serialize or deserialize objects stored in shared sessions, the classes of all objects stored in session attributes must be available to Web applications that share session data. For Web applications deployed on different containers, the classes may be placed in either the Web container or Web application classpath; however, for applications deployed in the same Web container, the classes must be placed in the Web container classpath. This is because most containers load each Web application using a separate ClassLoader.

---

**Note:** For advanced use cases where EAR cluster node-scoping or application server JVM cluster scoping is employed and you do not want session data shared across individual Web applications see “Preventing Web Applications from Sharing Session Data”.

---

Preventing Web Applications from Sharing Session Data

Sometimes you may want to explicitly prevent HTTP session data from being shared by different Java EE applications that participate in the same Coherence cluster. For example, assume you have two applications `HRPortal` and `InWeb` that share cached data in their EJB tiers but use different session data. In this case, it is desirable for both applications to be part of the same Coherence cluster, but undesirable for both applications to use the same clustered service for session data.

To prevent different Java EE applications from sharing session data, specify a unique session cache service name for each application:

1. Locate the `<service-name/>` parameters in each `session-cache-config.xml` file found in your application.
2. Set the parameters to a unique value for each application.
This forces each application to use a separate clustered service for session data.

3. Save the modified `session-cache-config.xml` files.

Example 4-1 illustrates a sample `session-cache-config.xml` file for an HRPortal application. To prevent the HRPortal application from sharing session data with the InWeb application, rename the `<service-name>` parameter for the replicated scheme to be `ReplicationSessionsMiscHRP`. Rename the `<service-name>` parameter for the distributed schemes to be `DistributedSessionsHRP`.

Example 4–1 Configuration to Prevent Applications from Sharing Session Data

```xml
<replicated-scheme>
  <scheme-name>default-replicated</scheme-name>
  <service-name>ReplicatedSessionsMisc</service-name> // rename this to ReplicatedSessionsMiscHRP
  <backing-map-scheme>
    <class-scheme>
      <scheme-ref>default-backing-map</scheme-ref>
    </class-scheme>
  </backing-map-scheme>
</replicated-scheme>

<distributed-scheme>
  <scheme-name>session-distributed</scheme-name>
  <service-name>DistributedSessions</service-name> // rename this to DistributedSessionsHRP
  <lease-granularity>member</lease-granularity>
  <backing-map-scheme>
    <class-scheme>
      <scheme-ref>default-backing-map</scheme-ref>
    </class-scheme>
  </backing-map-scheme>
</distributed-scheme>

<distributed-scheme>
  <scheme-name>session-certificate</scheme-name>
  <service-name>DistributedSessions</service-name> // rename this to DistributedSessionsHRP
  <lease-granularity>member</lease-granularity>
  <backing-map-scheme>
    <local-scheme>
      <scheme-ref>session-certificate-autoexpiring</scheme-ref>
    </local-scheme>
  </backing-map-scheme>
</distributed-scheme>
```

Keeping Session Cookies Separate

If you are using cookies to store session IDs, you must ensure that session cookies created by one application are not propagated to another application. To do this, you must set each application's session cookie domain and path in their `web.xml` file. The `context parameter coherence-session-cookie-path` sets the context path for a Web application. To prevent cookies from being propagated, ensure that no two applications share the same context path.

For example, assume you have two Web applications registered under the contexts paths `/web/HRPortal` and `/web/InWeb. To prevent the Web applications from sharing session data through cookies, set the `coherence-session-cookie-path`
parameter in one application’s web.xml file to /web/HRPortal; set the parameter in the other application’s web.xml file to /web/InWeb.

If your applications are deployed on different Web containers running on separate machines, then you can set the context parameter coherence-session-cookie-domain to ensure that they are not in the same domain.

For example, assume you have two Web applications running on server1.mydomain.com and server2.mydomain.com. To prevent session cookies from being shared between them, then set the coherence-session-cookie-domain parameter in one application’s web.xml file to server1.mydomain.com; set the parameter in the other application’s web.xml file to server2.mydomain.com.

### Session Attribute Scoping

In the case where sessions are shared across Web applications there are many instances where the application may want to scope individual session attributes so that they are either globally visible (that is, all Web applications can see and modify these attributes) or scoped to an individual Web application (that is, not visible to any instance of another application).

Coherence*Web provides the ability to control this behavior by using the AttributeScopeController interface. This optional interface is used to selectively scope attributes in cases when a session may be shared across multiple applications. This enables different applications to potentially use the same attribute names for application-scope state without accidentally reading, updating, or removing other applications’ attributes. In addition to having application-scoped information in the session, it allows the session to contain global (unscoped) information that is readable, updatable, and removable by any of the applications that share the session.

There are two implementations of this interface available out of the box: the ApplicationScopeController and the GlobalScopeController.

---

**Note:** After a configured AttributeScopeController is created, it is initialized with the name of the Web application, which it can use to qualify attribute names. You can configure the name of your Web application by using the display-name XML element in the Web application’s web.xml file.

---

### Cluster Node Isolation

When using Coherence*Web there are many deployment options to consider, one of which is the concept of cluster node isolation.

This option determines:

- The number of Coherence nodes that are created within an application server JVM.
- Where the Coherence library is deployed.

Applications can be application server-scoped, EAR-scoped, or WAR-scoped. This section describes these options. For detailed information on the XML configuration for each of these options, see “Packaging Applications and Configuring Cluster Nodes” on page 2-4.
Application Server-Scoped Cluster Nodes

With this configuration, all deployed applications in a container using Coherence*Web become part of one Coherence node. This configuration produces the smallest number of Coherence nodes in the cluster (one for each Web container JVM) and since the Coherence library (coherence.jar) is deployed in the container's classpath, only one copy of the Coherence classes is loaded into the JVM. This minimizes the use of resources. On the other hand, since all applications are using the same cluster node, all applications are affected if one application misbehaves.

Figure 4–5  Application Server-Scoped Cluster

Requirements for using this configuration are:

- Each deployed application must use the same version of Coherence and participate in the same cluster.
- Objects placed in the HTTP session must have their classes in the container's classpath.

"Packaging and Configuring Application Server-Scoped Cluster Nodes" on page 2-5 describes the XML configuration for application server-scoped cluster nodes.
Note: The application server-scoped cluster node configuration should be considered very carefully and *never* used in environments where the interaction between applications is unknown or unpredictable.

An example of such an environment may be a deployment where multiple application groups are deploying applications written independently, without carefully coordinating and enforcing their conventions and naming standards. With this configuration, all applications are part of the same cluster and the likelihood of collisions between namespaces for caches, services and other configuration settings is quite high and may lead to unexpected results.

For these reasons, Oracle Coherence strongly recommends that you use EAR-scoped and WAR-scoped cluster node configurations. If you are in doubt regarding which deployment topology to choose, or if this warning applies to your deployment, then *do not* choose the application server-scoped cluster node configuration.

**EAR-Scoped Cluster Nodes**

With this configuration, all deployed applications within each EAR become part of one Coherence node. This configuration produces the next smallest number of Coherence nodes in the cluster (one for each deployed EAR that uses Coherence*Web). Since the Coherence library (`coherence.jar`) is deployed in the application’s classpath, only one copy of the Coherence classes is loaded for each EAR. Since all Web applications in the EAR use the same cluster node, all Web applications in the EAR are affected if one of the Web applications misbehaves.

*Figure 4–6  EAR-Scoped Cluster*
EAR-scoped cluster nodes reduce the deployment effort as no changes to the application server classpath are required. This option is also ideal if you plan on deploying only one EAR to an application server.

Requirements for using this configuration are:

- The Coherence library (coherence.jar) must be deployed as part of the EAR file and listed as a Java module in META-INF/application.xml.
- Objects placed into the HTTP session must have their classes deployed as a Java EAR module in a similar fashion.

"Packaging and Configuring EAR-Scoped Cluster Nodes" on page 2-6 describes the XML configuration for EAR-scoped cluster nodes.

**WAR-Scoped Cluster Nodes**

With this configuration, each deployed Web application becomes its own Coherence node. This configuration produces the largest number of Coherence nodes in the cluster (one for each deployed WAR that uses Coherence*Web) and since the Coherence library (coherence.jar) is deployed in the Web application’s classpath, there will be as many copies of the Coherence classes loaded as there are deployed WARs. This results in the largest resource utilization out of the three options. However, since each deployed Web application is its own cluster node, Web applications are completely isolated from other potentially misbehaving Web applications.

WAR scoped cluster nodes reduce the deployment effort as no changes to the application server classpath are required. This option is also ideal if you plan on deploying only one WAR to an application server.

**Figure 4–7  WAR-Scoped Clusters**

![Diagram of WAR-Scoped Clusters](image)

Requirements for using this configuration are:
The Coherence library (coherence.jar) must be deployed as part of the WAR file (usually in WEB-INF/lib).

Objects placed into the HTTP session must have their classes deployed as part of the WAR file (in WEB-INF/lib or WEB-INF/classes).

"Packaging and Configuring WAR-Scoped Cluster Nodes" on page 2-6 describes the XML configuration for WAR-scoped cluster nodes.

Session Locking Modes

Oracle Coherence provides these configuration options for concurrent access to HTTP sessions.

- **Optimistic Locking (Default)**—Allows concurrent access to a session by multiple threads in a single JVM or multiple JVMs while prohibiting concurrent modification.
- **Member Locking**—Allows concurrent access and modification of a session by multiple threads in the same JVM while prohibiting concurrent access by threads in different JVMs.
- **Application Locking**—Allows concurrent access and modification of a session by multiple threads in the same Web application instance while prohibiting concurrent access by threads in different Web application instances.
- **Thread Locking**—Prohibits concurrent access and modification of a session by multiple threads in a single JVM or multiple JVMs.

For more information on the parameters described in this section, see Appendix A, "Coherence*Web Configuration Parameters."

**Optimistic Locking (Default)**

The Optimistic Locking mode allows multiple Web container threads in one or more JVMs to access the same session concurrently. This setting does not use explicit locking; rather an optimistic approach is used to detect and prevent concurrent updates upon completion of an HTTP request that modifies the session. When Coherence*Web detects a concurrent modification, a `ConcurrentModificationException` is thrown to the application; therefore an application must be prepared to handle this exception in an appropriate manner.

This mode can be configured by setting the `coherence-session-member-locking` parameter to false.

**Member Locking**

The Member Locking mode allows multiple Web container threads in the same JVM to access and modify the same session concurrently, but prohibits concurrent access by threads in different JVMs. This is accomplished by acquiring a member-level lock for an HTTP session at the beginning of a request and releasing the lock upon completion of the request. For more information on member-level locks, see `<lease-granularity>` in the `distributed-scheme` section of the Developer’s Guide for Oracle Coherence.

This mode can be configured by setting the `coherence-session-member-locking` parameter to true.
Application Locking

The Application Locking mode restricts access (and modification) to a session to threads in a single Web application instance at a time. This is accomplished by acquiring both a member-level and application-level lock for an HTTP session at the beginning of a request and releasing both locks upon completion of the request. For more information on member-level locks, see `<lease-granularity>` in the distributed-scheme section of the Developer's Guide for Oracle Coherence.

This mode can be configured by setting the `coherence-session-app-locking` parameter to `true`. Note that setting this to `true` will imply a setting of `true` for `coherence-session-member-locking`.

Thread Locking

The Thread Locking mode restricts access (and modification) to a session to a single thread in a single JVM at a time. This is accomplished by acquiring both a member level, application level, and thread-level lock for an HTTP session at the beginning of a request and releasing all three locks upon completion of the request. For more information on member-level locks, see `<lease-granularity>` in the distributed-scheme section of the Developer's Guide for Oracle Coherence.

This mode can be configured by setting the `coherence-session-thread-locking` parameter to `true`. Note that setting this to `true` implies a setting of `true` for both `coherence-session-member-locking` and `coherence-session-app-locking`.

Using Locking in HTTP Sessions

Enabling Member, Application, or Thread Locking for HTTP session access indicates that Coherence*Web will acquire a cluster-wide lock for every HTTP request that requires access to a session; the exception to this is when sticky load balancing is available and the Coherence*Web sticky session optimization is enabled. By default, threads that attempt to access a locked session (locked by a thread in a different JVM) block until the lock can be acquired. If you want to enable a timeout for lock acquisition, you can configure it by using the `tangosol.coherence.servlet.lock.timeout` system property in the container's startup script (for example `-Dtangosol.coherence.servlet.lock.timeout=30s`).

Many Web applications do not have such a strict concurrency requirement. For these applications, using the Optimistic Locking mode has the following advantages:

- The overhead of obtaining and releasing cluster wide locks for every HTTP request is eliminated.
- Requests can be load balanced away from failing or unresponsive JVMs to healthy JVMs without requiring the unresponsive JVM to release the cluster-wide lock on the session.

Enabling Sticky Session Optimizations

If Member, Application, or Thread Locking is a requirement for a Web application that resides behind a sticky load balancer, Coherence*Web provides an optimization for obtaining the cluster-wide lock required for HTTP session access. By definition, a sticky load balancer attempts to route each request for a given session to the same application server JVM that it previously routed requests to for that same session, which initially is the application server JVM that created the session. The sticky session optimizations takes advantage of this behavior by retaining the cluster-wide lock for a
session until the session expires or until it is asked to release it. If, for whatever reason, the sticky load balancer sends a request for the same session to another application server JVM, that JVM will ask the JVM that owns the lock on the session to release the lock as soon as possible. This is implemented using an invocation service. For more information, see the SessionOwnership entry in Table B–2.

Sticky session optimization can be enabled by setting the coherence-sticky-sessions parameter to true.

Deployment Topologies

Coherence*Web supports most of the same deployment topologies that Coherence does including in-process, out-of-process (that is, client/server deployment), and bridging clients and servers over Coherence*Extend. The major supported deployment topologies are described in the following sections.

- In-Process
- Out-of-Process
- Out-of-Process with Coherence*Extend

In-Process

The In-Process topology is not recommended for production use. This topology is supported mainly for development and testing. By storing the session data in-process with the application server, this topology is very easy to get up and running quickly for smoke tests, development and testing.

![In-Process Deployment Topology](image)

Out-of-Process

In the Out of Process deployment topology, the application servers (that is, application server tier) are configured as cache clients (that is, tangosol.coherence.distributed.localstorage=false) and there are dedicated JVMs running as cache servers, physically storing and managing the clustered data.

This approach has these benefits:

- Session data storage is off-loaded from the application server tier to the cache server tier. This reduces heap usage, garbage collection times, and so on.
- It allows for the two tiers to be scaled independently of one another. If more application processing power is needed, just start more application servers. If more session storage capacity is needed, just start more cache servers.
The Out-of-Process topology is the default recommendation of Oracle Coherence due to its flexibility.

**Figure 4–9 Out of Process Deployment Topology**

**Out-of-Process with Coherence*Extend**

The Out-of-Process with Coherence*Extend topology is similar to the Out-of-Process topology except that the communication between the application server tier and the cache server tier are over Coherence*Extend (TCP/IP). For information on configuring this scenario, see "Configuring Coherence*Web with Coherence*Extend" on page 4-24.

This approach has the same benefits as the Out-of-Process topology and the ability to segment deployment of application servers and cache servers. This is ideal in an environment where application servers are on a network that does not support UDP. The cache servers can be set up in a separate dedicated network, with the application servers connecting to the cluster by using TCP.
Managing and Monitoring Applications with JMX

**Note:** To enable Coherence*Web JMX Management and Monitoring, this section assumes that you have first set up the Coherence Clustered JMX Framework. To set up this framework, see the configuration and installation instructions in *How to Manage Coherence with JMX* in the *Developer’s Guide for Oracle Coherence*.

The management attributes and operations for Web applications that use Coherence*Web for HTTP session management are exposed through the HttpSessionManagerMBean interface (com.tangosol.coherence.servlet.management.HttpSessionManagerMBean).

During startup, each Coherence*Web Web application registers a single instance of HttpSessionManagerMBean. The MBean is unregistered when the Web application shuts down. *Table 4–1* describes the MBean’s object name used for registration.

**Table 4–1  Object Name for the HttpSessionManagerMBean**

<table>
<thead>
<tr>
<th>Managed Bean</th>
<th>Object Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>HttpSessionManagerMBean</td>
<td>type=HttpSessionManager, nodeId=cluster node id, appId=web application id</td>
</tr>
</tbody>
</table>

*Table 4–2* describes the information that is returned by the HttpSessionManagerMBean. All of the names represent attributes, except resetStatistics, which is an operation.

Several of the MBean attributes use the following prefixes:
- LocalSession—indicates a session that is not distributed to all members of the cluster. The session remains "local" to the originating server until a later point in the life of the session.
- LocalAttribute—indicates a session attribute that is not distributed to all members of the cluster.
- Overflow—typically, a larger and slower back-end cache that catches entries evicted from a faster front-end cache.

Table 4–2  Information Returned by the HttpSessionManagerMBean

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CollectionClassName</td>
<td>String</td>
<td>The fully qualified class name of the HttpSessionCollection implementation in use. The HttpSessionCollection interface is an abstract model for a collection of HttpSessionModel objects. The interface is not at all concerned with how the sessions are communicated between the clients and the servers.</td>
</tr>
<tr>
<td>FactoryClassName</td>
<td>String</td>
<td>The fully qualified class name of the Factory implementation in use. The SessionHelper.Factory is used by the SessionHelper to obtain objects that implement various important parts of the Servlet specification. It can be placed in front of the application in place of the application server's own objects, thus changing the &quot;apparent implementation&quot; of the application server itself (for example, adding clustering.)</td>
</tr>
<tr>
<td>LocalAttributeCacheName</td>
<td>String</td>
<td>The name of the local cache that stores non-distributed session attributes. If the attribute displays null then local session attribute storage is disabled.</td>
</tr>
<tr>
<td>LocalAttributeCount</td>
<td>Integer</td>
<td>The number of non-distributed session attributes stored in the local session attribute cache. If the attribute displays -1, then local session attribute storage is disabled.</td>
</tr>
<tr>
<td>LocalSessionCacheName</td>
<td>String</td>
<td>The name of the local cache that stores non-distributed sessions. If the attribute displays null, then local session storage is disabled.</td>
</tr>
<tr>
<td>LocalSessionCount</td>
<td>Integer</td>
<td>The number of non-distributed sessions stored in the local session cache. If the attribute displays -1, then local session storage is disabled.</td>
</tr>
<tr>
<td>OverflowAverageSize</td>
<td>Integer</td>
<td>The average size (in bytes) of the session attributes stored in the &quot;overflow&quot; clustered cache since the last time statistics were reset. If the attribute displays -1, then a SplitHttpSessionCollection is not in use.</td>
</tr>
<tr>
<td>OverflowCacheName</td>
<td>String</td>
<td>The name of the clustered cache that stores the &quot;large attributes&quot; that exceed a certain size and thus are determined to be more efficiently managed as separate cache entries and not as part of the serialized session object itself. Null is displayed if a SplitHttpSessionCollection is not in use.</td>
</tr>
<tr>
<td>OverflowMaxSize</td>
<td>Integer</td>
<td>The maximum size (in bytes) of a session attribute stored in the &quot;overflow&quot; clustered cache since the last time statistics were reset. The attribute displays -1 if a SplitHttpSessionCollection is not in use.</td>
</tr>
<tr>
<td>OverflowThreshold</td>
<td>Integer</td>
<td>The minimum length (in bytes) that the serialized form of an attribute value must be for that attribute value to be stored in the separate &quot;overflow&quot; cache that is reserved for large attributes. The attribute displays -1 if a SplitHttpSessionCollection is not in use.</td>
</tr>
<tr>
<td>OverflowUpdates</td>
<td>Integer</td>
<td>The number of updates to session attributes stored in the &quot;overflow&quot; clustered cache since the last time statistics were reset. The attribute displays -1 if a SplitHttpSessionCollection is not in use.</td>
</tr>
</tbody>
</table>
**Table 4–2  (Cont.) Information Returned by the HttpSessionManagerMBean**

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SessionAverageLifetime</td>
<td>Integer</td>
<td>The average lifetime (in seconds) of session objects invalidated (either due to expiration or to an explicit invalidation) since the last time statistics were reset.</td>
</tr>
<tr>
<td>SessionAverageSize</td>
<td>Integer</td>
<td>The average size (in bytes) of session objects placed in the session storage clustered cache since the last time statistics were reset.</td>
</tr>
<tr>
<td>SessionCacheName</td>
<td>String</td>
<td>The name of the clustered cache that stores serialized session objects.</td>
</tr>
<tr>
<td>SessionIdLength</td>
<td>Integer</td>
<td>The length (in characters) of generated session IDs.</td>
</tr>
<tr>
<td>SessionMaxSize</td>
<td>Integer</td>
<td>The maximum size (in bytes) of a session object placed in the session storage clustered cache since the last time statistics were reset.</td>
</tr>
<tr>
<td>SessionMinSize</td>
<td>Integer</td>
<td>The minimum size (in bytes) of a session object placed in the session storage clustered cache since the last time statistics were reset.</td>
</tr>
<tr>
<td>SessionStickyCount</td>
<td>Integer</td>
<td>The number of session objects that are pinned to this instance of the Web application. The attribute displays -1 if sticky session optimizations are disabled.</td>
</tr>
<tr>
<td>SessionTimeout</td>
<td>Integer</td>
<td>The session expiration time (in seconds). The attribute displays -1 if sessions never expire.</td>
</tr>
<tr>
<td>SessionUpdates</td>
<td>Integer</td>
<td>The number of updates of session object stored in the session storage clustered cache since the last time statistics were reset.</td>
</tr>
<tr>
<td>ServletContextCacheName</td>
<td>String</td>
<td>The name of the clustered cache that stores javax.servlet.ServletContext attributes. The attribute displays null if the ServletContext is not clustered.</td>
</tr>
<tr>
<td>ServletContextName</td>
<td>String</td>
<td>The name of the Web application ServletContext.</td>
</tr>
<tr>
<td>resetStatistics (operation)</td>
<td>void</td>
<td>Reset the session management statistics.</td>
</tr>
</tbody>
</table>

*Figure 4–11* illustrates the HttpSessionManagerMBean as it is displayed in the JConsole browser.
Cleaning Up Expired HTTP Sessions

As part of Coherence*Web Session Management Module, HTTP sessions are eventually cleaned up by the Session Reaper, and the associated memory is freed. The Session Reaper provides a service similar to the JVM's own Garbage Collection (GC) capability: the Session Reaper is responsible for destroying any session that is no longer used, which is determined when that session has timed out.

Each HTTP session contains two pieces of information that determine when it has timed out. The first is the `LastAccessedTime` property of the session, which is the timestamp of the most recent activity involving the session. The second is the `MaxInactiveInterval` property of the session, which specifies how long the session is kept alive without any activity; a typical value for this property is 30 minutes. The `MaxInactiveInterval` property defaults to the value specified for the `coherence-session-expire-seconds` configuration option, but it can be modified on a session-by-session basis.

Each time that an HTTP request is received by the server, if there is an HTTP session associated with that request, then the `LastAccessedTime` property of the session is automatically updated to the current time. As long as requests continue to arrive related to that session, it is kept alive, but when a period of inactivity occurs longer than that specified by the `MaxInactiveInterval` property, then the session expires. Session expiration is passive—occurring only due to the passing of time. The Coherence*Web Session Reaper scans for sessions that have expired, and when it finds expired sessions it cleans them up.
Understanding the Session Reaper

The Session Reaper configuration answers three basic questions:

- On which servers will the Reaper run?
- How frequently will the Reaper run?
- When the Reaper runs, on which servers will it look for expired sessions?

The Session Reaper runs as part of the application server. That means that if Coherence is configured to provide a separate cache tier (made up of "cache servers"), then the Session Reaper does not run on those cache servers.

Consider the three different topologies used with Coherence*Web:

- **In-Process**—The application servers that run Coherence*Web are storage-enabled, so that the HTTP session storage is co-located with the application servers. No separate cache servers are used for HTTP session storage.

- **Out-of-Process**—The application servers that run Coherence*Web are storage-disabled members of the Coherence cluster. Separate cache servers are used for HTTP session storage.

- **Out-of-Process with Coherence*Extend**—The application servers that run Coherence*Web are not part of a Coherence cluster; the application servers use Coherence*Extend to attach to a Coherence cluster which contains cache servers used for HTTP session storage.

Every application server running Coherence*Web runs the Session Reaper. By default, the Session Reaper runs concurrently on all of the application servers, so that all of the servers share the workload of identifying and cleaning up expired sessions. The `coherence-reaperdaemon-cluster-coordinated` configuration option causes the cluster to coordinate reaping so that only one server at a time is performing the actual reaping; the use of this option is not suggested, and it cannot be used with the Coherence*Web over Coherence*Extend topology.

The Session Reaper is configured to scan the entire set of sessions over a certain period, called a reaping cycle, which defaults to five minutes. This length of the reaping cycle is specified by the `coherence-reaperdaemon-cycle-seconds` option. Since the Session Reaper is expected to scan all of the sessions that it is responsible for and to clean up any expired sessions within the reaping cycle, this setting indicates to the Session Reaper how aggressively it must work. If the cycle length is configured too short, the Session Reaper uses additional resources without providing additional benefit. If the cycle length is configured too long, then sessions may not be cleaned up as quickly after they have expired. In most situations, it is far preferable to reduce resource usage than to ensure that sessions are cleaned up quickly after they expire. Consequently, the default cycle of five minutes is a good balance between promptness of cleanup and minimal resource usage.

During the reaping cycle, the Session Reaper scans for expired sessions. In most cases, the Session Reaper takes responsibility for scanning all of the HTTP sessions across the entire cluster, but there is an optimization available for the Single Tier topology. In the Single Tier topology, when all of the sessions are being managed by storage-enabled Coherence cluster members that are also running the application server, the session storage is co-located with the application server. Consequently, it is possible for the Session Reaper on each application server to only scan the sessions that are stored locally. This behavior can be enabled by setting the `coherence-reaperdaemon-assume-locality` configuration option to true.
Regardless of whether the Session Reaper scans only co-located sessions or all sessions, it does so in a very efficient manner by using these advanced capabilities of the Coherence data grid:

- Starting with the current version of Coherence, the Session Reaper does not actually look at each session; instead, it delegates the search for expired sessions to the data grid using a custom ValueExtractor implementation. This ValueExtractor takes advantage of the BinaryEntry interface introduced in Coherence version 3.5 so that it can determine if the session has expired without even deserializing the session. As a result, the selection of expired sessions can be delegated to the data grid just like any other parallel query, and can be executed by storage-enabled Coherence members in a very efficient manner.

- Instead of selecting all of the expired sessions immediately using a parallel query, the Session Reaper only queries one member at a time; this allows the Session Reaper to divide the work of the query across the duration of the reaping cycle. Additionally, this eliminates the need for group communication when querying for expired sessions.

- Since the work of cleaning up expired sessions is broken up across the entire reaping cycle, this ensures that the selection of expired sessions is also broken up across the reaping cycle, so that the selection occurs close before the clean-up of expired sessions, thus reducing the chance that multiple application servers would attempt to clean up the same expired sessions. The Session Reaper uses the com.tangosol.net.partition.PartitionedIterator class to automatically query on a member-by-member basis, and in a random order that avoids harmonics in large-scale clusters.

Each storage-enabled member can very efficiently scan for any expired sessions, and it only has to scan one time per application server per reaper cycle. The result is an out-of-the-box Session Reaper configuration that works well for application server clusters with only two servers, and application server clusters with several hundred servers. Furthermore, the configuration works well for applications with several hundred concurrent sessions, and for applications with several million concurrent sessions.

To ensure that the Session Reaper does not impact the smooth operation of the application server, it breaks up its work into chunks and schedules that work in a manner that spreads the work across the entire reaping cycle. Since the Session Reaper has to know how much work it must schedule, it maintains statistics on the amount of work that it performed in previous cycles, and uses statistical weighting to ensure that statistics from recent reaping cycles count more heavily. There are several reasons why the Session Reaper breaks up the work in this manner:

- If the Session Reaper consumed a large number of CPU cycles at one time, it could cause the application to be less responsive to users. By doing a small portion of the work at a time, the application remains responsive.

- One of the key performance enablers for Coherence*Web is the near caching feature of Coherence; since the sessions that are expired are accessed through that same near cache to clean them, expiring too many sessions too quickly could cause the cache to evict sessions that are being used on that application server, leading to performance loss.

The Session Reaper performs its job efficiently, even with the default out-of-the-box configuration by:

- delegating as much work as possible to the data grid
- delegating work to only one member at a time
Overriding the Distribution of HTTP Sessions and Attributes

- avoiding group communication
- enabling the data grid to find expired sessions without even deserializing them
- restricting the usage of CPU cycles
- avoiding cache-thrashing of the near caches that Coherence*Web relies on for performance

Configuring the Session Reaper

The following list contains suggestions for tuning the out-of-the-box configuration of the Session Reaper:

- If the application is deployed with the in-process topology, then set the `coherence-reaperdaemon-assume-locality` configuration option to `true`.
- Since all of the application servers are responsible for scanning for expired sessions, it is reasonable to increase the `coherence-reaperdaemon-cycle-seconds` configuration option if the cluster is larger than ten application servers. The larger the number of application servers, the longer the cycle can be; for example, with 200 servers, it would be reasonable to set the length of the reaper cycle as high as 30 minutes (that is, setting the `coherence-reaperdaemon-cycle-seconds` configuration option to 1800).

Overriding the Distribution of HTTP Sessions and Attributes

The Coherence*Web Session Distribution Controller, described by the `HttpSessionCollection.SessionDistributionController` interface, enables you to override the default distribution of HTTP sessions and attributes in a Web application. An implementation of the `SessionDistributionController` interface can mark sessions and/or attributes in either of the following ways:

- local—a `local` session and/or attribute is stored on the originating server's heap, and thus, only accessible by that server
- distributed—a `distributed` session and/or attribute is stored within the Coherence grid, and thus, accessible to other server JVMs

At any point during the life of a session, the session and/or attributes for that session can be transitioned from local or distributed. However, once a session and/or attribute is distributed it cannot transition back to local.

You can use the Session Distribution Controller in any of the following ways:

- You can allow new sessions to remain "local" until you add an attribute (for example, when you add the first item to an on-line shopping cart); the idea being that a session only needs to be fault-tolerant when it contains valuable data.
- Some Web frameworks use session attributes to store UI rendering state. Often, this data cannot be distributed because it is not serializable. Using the Session Distribution Controller, these attributes can be kept local while allowing the rest of the session attributes to be distributed.
- The Session Distribution Controller can assist in the conversion from non-distributed to distributed systems, especially when the cost of distributing all sessions and all attributes is a consideration.
Implementing a Session Distribution Controller

Example 4–2 illustrates a sample implementation of the HttpSessionCollection.SessionDistributionController interface. In the sample, sessions are tested as to whether they have a shopping cart attached (only these sessions will be distributed). Next, the session is tested whether it contains a certain attribute. If the attribute is found to be present, then it is not distributed.

Example 4–2 Sample Session Distribution Controller Implementation

```java
import com.tangosol.coherence.servlet.HttpSessionCollection;
import com.tangosol.coherence.servlet.HttpSessionModel;

/**<n
* Sample implementation of SessionDistributionController
*/

public class CustomSessionDistributionController
    implements HttpSessionCollection.SessionDistributionController
{
    public void init(HttpSessionCollection collection)
    {
    }

    /**
     * Only distribute sessions that have a shopping cart.
     * @param model Coherence representation of the HTTP session
     * @return true if the session should be distributed
     */
    public boolean isSessionDistributed(HttpSessionModel model)
    {
        return model.getAttribute("shopping-cart") != null;
    }

    /**
     * If a session is "distributed", then distribute all attributes with the
     * exception of the "ui-rendering" attribute.
     * @param model Coherence representation of the HTTP session
     * @param sName name of the attribute to check
     * @return true if the attribute should be distributed
     */
    public boolean isSessionAttributeDistributed(HttpSessionModel model, String sName)
    {
        return !"ui-rendering".equals(sName);
    }
}
```

Registering a Session Distribution Controller Implementation

Once you have written your SessionDistributionController implementation, you can register it with your application by using the coherence-distributioncontroller-class configuration parameter. Note that to use the Session Distribution Controller, you must also enable the coherence-sticky-sessions parameter. Appendix A, "Coherence*Web Configuration Parameters" provides more information on these parameters.
Configuring Coherence*Web with Coherence*Extend

One of the deployment options for Coherence*Web is to use Coherence*Extend to connect Web container JVMs to the cluster by using TCP/IP. This configuration should be considered if any of the following situations applies:

- The Web tier JVMs are in a DMZ while the Coherence cluster is behind a firewall.
- The Web tier is in an environment that does not support UDP.
- Web tier JVMs experience long and/or frequent GC pauses.
- Web tier JVMs are restarted frequently.

In this type of deployment, there are three types of participants:

- Web tier JVMs—These are the Extend clients in this topology. They are not members of the cluster; instead, they connect to a proxy node in the cluster that will issue requests to the cluster on their behalf.
- Proxy JVMs—These nodes are storage-disabled members of the cluster that accept and manage TCP/IP connections from Extend clients. Requests that arrive from clients will be sent into the cluster, and responses will be sent back through the TCP/IP connections.
- Storage JVMs—These JVMs are used to store the actual session data in memory.

These are the general steps to configure Coherence*Web to use Coherence*Extend:

1. Configure Coherence*Web to use the Optimistic Locking mode (see "Optimistic Locking (Default)" on page 4-12).
2. Configure a cache configuration file for the proxy and storage JVMs
3. Modify the Web tier cache configuration file to point to one or more of the proxy JVMs

The following sections describe these steps in more detail.

Configuring Coherence*Web for Optimistic Locking

To enable the Optimistic Locking mode for your Web application, make sure the Coherence*Web configuration parameters in Table 4–3 are set to the specified values.

Table 4–3  Coherence*Web Parameter Settings for Optimistic Locking

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>coherence-session-member-locking</td>
<td>false</td>
</tr>
<tr>
<td>coherence-sticky-sessions</td>
<td>false</td>
</tr>
<tr>
<td>coherence-preserve-attributes</td>
<td>false</td>
</tr>
</tbody>
</table>

See Appendix A, "Coherence*Web Configuration Parameters" for more information on these parameters.

Configuring the Cache for Proxy and Storage JVMs

The session cache configuration file (WEB-INF/classes/session-cache-config.xml) is an example Coherence*Web cache configuration file that uses Coherence*Extend.
This session cache configuration file should be used for the proxy and server JVMs. It contains system property overrides that allow the same file to be used for both proxy and storage JVMs. When used by a proxy JVM, the system properties described in Table 4–4 should be specified.

**Table 4–4  System Property Values for Proxy JVMs**

<table>
<thead>
<tr>
<th>System Property Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tangosol.coherence.session.localstorage</td>
<td>false</td>
</tr>
<tr>
<td>tangosol.coherence.session.proxy</td>
<td>true</td>
</tr>
<tr>
<td>tangosol.coherence.session.proxy.localhost</td>
<td>the host name or IP address of the NIC the proxy will bind to</td>
</tr>
<tr>
<td>tangosol.coherence.session.proxy.localport</td>
<td>a unique port number the proxy will bind to</td>
</tr>
</tbody>
</table>

When used by a storage JVM, the system properties described in Table 4–5 should be specified.

**Table 4–5  System Property Values for Storage JVMs**

<table>
<thead>
<tr>
<th>System Property Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tangosol.coherence.session.localstorage</td>
<td>true</td>
</tr>
<tr>
<td>tangosol.coherence.session.proxy</td>
<td>false</td>
</tr>
</tbody>
</table>

Example 4–3 illustrates the complete server-side session cache configuration file.

**Example 4–3  session-cache-config-server.xml File**

```xml
<?xml version="1.0"?>
<!DOCTYPE cache-config SYSTEM "cache-config.dtd">
<!-- Server-side cache configuration descriptor for Coherence*Web over Coherence*Extend (see session-cache-config-client.xml). -->
<!-- The clustered cache used to store Session management data. -->
<cache-mapping>
  <cache-name>session-management</cache-name>
  <scheme-name>session-distributed</scheme-name>
</cache-mapping>

<!-- The clustered cache used to store ServletContext attributes. -->
<cache-mapping>
  <cache-name>servletcontext-storage</cache-name>
  <scheme-name>session-distributed</scheme-name>
</cache-mapping>

<!-- The clustered cache used to store Session attributes. -->
```
<cache-mapping>
  <cache-name>session-storage</cache-name>
  <scheme-name>session-distributed</scheme-name>
</cache-mapping>

<!--
The clustered cache used to store the "overflowing" (split-out due to size)
Session attributes. Only used for the "Split" model.
-->
<cache-mapping>
  <cache-name>session-overflow</cache-name>
  <scheme-name>session-distributed</scheme-name>
</cache-mapping>

<!--
The clustered cache used to store IDs of "recently departed" Sessions.
-->
<cache-mapping>
  <cache-name>session-death-certificates</cache-name>
  <scheme-name>session-certificate</scheme-name>
</cache-mapping>
</caching-scheme-mapping>

<caching-schemes>
  <!--
  Distributed caching scheme used by the various Session caches.
  -->
  <distributed-scheme>
    <scheme-name>session-distributed</scheme-name>
    <scheme-ref>session-base</scheme-ref>
    <backing-map-scheme>
      <local-scheme>
        <scheme-ref>unlimited-local</scheme-ref>
      </local-scheme>
    </backing-map-scheme>
  </distributed-scheme>

  <!--
  Distributed caching scheme used by the "recently departed" Session cache.
  -->
  <distributed-scheme>
    <scheme-name>session-certificate</scheme-name>
    <scheme-ref>session-base</scheme-ref>
    <backing-map-scheme>
      <local-scheme>
        <eviction-policy>HYBRID</eviction-policy>
        <high-units>4000</high-units>
        <low-units>3000</low-units>
        <expiry-delay>86400</expiry-delay>
      </local-scheme>
    </backing-map-scheme>
  </distributed-scheme>
</caching-schemes>

<!--
"Base" Distributed caching scheme that defines common configuration.
-->
<distributed-scheme>
  <scheme-name>session-base</scheme-name>
  <service-name>DistributedSessions</service-name>
  <serializer>
    <class-name>com.tangosol.io.DefaultSerializer</class-name>
  </serializer>
  <thread-count>0</thread-count>
  <lease-granularity>member</lease-granularity>
  <local-storage system-property="tangosol.coherence.session.localstorage">true</local-storage>
  <partition-count>257</partition-count>
  <backup-count>1</backup-count>
  <backup-storage>
    <type>on-heap</type>
  </backup-storage>
  <backing-map-scheme>
    <local-scheme>
      <scheme-ref>unlimited-local</scheme-ref>
      <local-scheme>
        <backing-map-scheme>
          <local-scheme>
            <scheme-ref>unlimited-local</scheme-ref>
          </local-scheme>
        </backing-map-scheme>
      </local-scheme>
    </local-scheme>
  </backing-map-scheme>
  <autostart>true</autostart>
</distributed-scheme>

<!-- Proxy scheme that Coherence*Web clients used to connect to the cluster. -->
<proxy-scheme>
  <service-name>SessionProxy</service-name>
  <thread-count>10</thread-count>
  <acceptor-config>
    <serializer>
      <class-name>com.tangosol.io.DefaultSerializer</class-name>
    </serializer>
    <tcp-acceptor>
      <local-address>
        <address system-property="tangosol.coherence.session.proxy.localhost">localhost</address>
        <port system-property="tangosol.coherence.session.proxy.localport">9099</port>
      </local-address>
    </tcp-acceptor>
  </acceptor-config>
  <autostart system-property="tangosol.coherence.session.proxy">false</autostart>
</proxy-scheme>

<!-- Local caching scheme definition used by all caches that do not require an eviction policy. -->
<local-scheme>
  <scheme-name>unlimited-local</scheme-name>
  <service-name>LocalSessionCache</service-name>
</local-scheme>
Configuring the Cache for Web Tier JVMs

The session-cache-config-client.xml file illustrated in Example 4-4 is an example Coherence*Web cache configuration file that uses Coherence*Extend. This cache configuration file should be used by the Web tier JVMs. To use and install this file, follow these steps:

1. Add proxy JVM hostnames/IP addresses and ports to the <remote-addresses/> section of the file. In most cases, you should include the hostname/IP address and port of all proxy JVMs for load balancing and failover.

   **Note:** The <remote-addresses> element contains the proxy server(s) that the Web container will connect to. By default, the Web container will pick an address at random (assuming that there is more than one address in the configuration.) If the connection between the Web container and the proxy is broken, the container will connect to another proxy in the list.

2. Rename the file to session-cache-config.xml.

3. Place the file in the WEB-INF/classes directory of your Web application. If you used the WebInstaller to install Coherence*Web, replace the existing file that was added by the WebInstaller.

Example 4-4 illustrates the complete client-side session cache configuration file.

**Example 4-4  session-cache-config-client.xml File**

```xml
<?xml version="1.0"?>
<!DOCTYPE cache-config SYSTEM "cache-config.dtd">
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!--                                                                       -->
<!--     Client-side cache configuration descriptor for Coherence*Web over  -->
<!--     Coherence*Extend (see session-cache-config-server.xml).           -->
<!--                                                                       -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<cache-config>
  <caching-scheme-mapping>
    <!-- The clustered cache used to store Session management data. -->
    <cache-mapping>
      <cache-name>session-management</cache-name>
      <scheme-name>session-near</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>
  <caching-scheme-mapping>
    <!-- The clustered cache used to store ServletContext attributes. -->
    <cache-mapping>
      <cache-name>servletcontext-storage</cache-name>
      <scheme-name>session-near</scheme-name>
    </cache-mapping>
</cache-config>
```
The clustered cache used to store Session attributes.

The clustered cache used to store the "overflowing" (split-out due to size) Session attributes. Only used for the "Split" model.

The clustered cache used to store IDs of "recently departed" Sessions.

Near caching scheme used by the Session attribute cache. The front cache uses a Local caching scheme and the back cache uses a Remote caching scheme.
<remote-cache-scheme>
  <scheme-name>session-remote</scheme-name>
  <initiator-config>
    <serializer>
      <class-name>com.tangosol.io.DefaultSerializer</class-name>
    </serializer>
    <tcp-initiator>
      <remote-addresses>
        <!--
        The following list of addresses should include the hostname and port
        of all running proxy JVMs. This is for both load balancing and
        failover of requests from the Web tier.
        -->
        <socket-address>
          <address>localhost</address>
          <port>9099</port>
        </socket-address>
      </remote-addresses>
    </tcp-initiator>
  </initiator-config>
</remote-cache-scheme>
</caching-schemes>
</cache-config>
Installing Coherence*Web on WebLogic Portal

Coherence*Web can be installed in a WebLogic Portal environment to provide session state management based on Coherence. Coherence*Web allows for more advanced deployment models, session models, and locking modes in a clustered environment. For more information on these features, see Chapter 4, "Coherence*Web Session Management Features."

Installing Coherence*Web with WebLogic Portal

- **Note:** As with other WebLogic Server applications, all of the information and instructions in Chapter 2, "Installing Coherence*Web on WebLogic Server 9.2 MP1 and 10.3" apply to deploying Coherence*Web with WebLogic Portal. Please read Chapter 2 before proceeding. It provides much needed context for the following information.

Complete these steps to install Coherence*Web with WebLogic Portal applications:

1. Install the appropriate WebLogic Patch (ID 6W2W for WLS 10.3, or ID AJQB for WLS 9.2 MP1) and start a cache server using steps 1-3 in "Overview Of Configuration and Deployment" on page 2-2.

2. Copy the coherence.jar (included in the Coherence distribution) into the APP-INF\lib directory of the portal enterprise application.

3. (Optional) If you want to use the Coherence P13N CacheProvider, then copy the coherence-wlp.jar into the APP-INF\lib of the portal enterprise application.

   See the Integration Guide for Oracle Coherence for more information on the P13N CacheProvider SPI implementation and WSRP-federated portals.

4. (Optional) If you want to use the Coherence P13N Cache as the default cache provider, add the following element before the first <cache> element to the META-INF\p13n-cache-config.xml file in the portal enterprise application:

   `<default-provider-id>com.tangosol.coherence.weblogic</default-provider-id>`

5. Reference coherence-web-spi.war by using a library-reference in the WEB-INF\weblogic.xml file in the portal Web application:

   `<wls:library-ref>`
6. To enable Coherence*Web sessions, set the application parameter `coherence-web-sessions-enabled` to `true` in the `WEB-INF/web.xml` file in the portal Web application:

   ```xml
   <context-param>
     <param-name>coherence-web-sessions-enabled</param-name>
     <param-value>true</param-value>
   </context-param>
   ```

7. Create a `.EAR` of the test application using workshop’s EAR deployment. This EAR will be deployed to the cluster for testing.

8. In the portal domain, first deploy `coherence-web-spi.war` as a library to the cluster.

9. In the portal domain, deploy the application EAR to the cluster.
Coherence*Web provides a wide variety of configuration options as described in Table A–1. The process for configuring Coherence*Web is slightly different between the WebLogic SPI-based installation case and the generic WebInstaller installation case.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>coherence-factory-class</td>
<td>The fully qualified class name of the SessionHelper.Factory to use. Defaults to com.tangosol.coherence.servlet.api.XX. DefaultFactory where XX is 22, 23, 24, or 25 for Servlet 2.2, 2.3, 2.4, or 2.5 containers respectively.</td>
</tr>
<tr>
<td>coherence-sessioncollection-class</td>
<td>The fully qualified class name of the HttpSessionCollection implementation to use. Possible values include:</td>
</tr>
<tr>
<td></td>
<td>■ com.tangosol.coherence.servlet.MonolithicHttpSessionCollection</td>
</tr>
<tr>
<td></td>
<td>■ com.tangosol.coherence.servlet.SplitHttpSessionCollection</td>
</tr>
<tr>
<td></td>
<td>■ com.tangosol.coherence.servlet.TraditionalHttpSessionCollection</td>
</tr>
<tr>
<td></td>
<td>A value for this parameter must be specified.</td>
</tr>
<tr>
<td>coherence-cluster-owned</td>
<td>If true, Coherence*Web automatically shuts down the Coherence node when the Web application shuts down. You must use the WAR scoped cluster node deployment model in this case. See &quot;WAR-Scoped Cluster Nodes&quot; on page 4-11 for more information. If false, the Web application is responsible for shutting down the Coherence node (see com.tangosol.net.CacheFactory.shutdown()) in accordance to its own considerations. You must carefully consider a cluster node scoping deployment model in this case and the circumstances under which the application shuts down the Coherence node and the side-effects of doing so. See &quot;Cluster Node Isolation&quot; on page 4-8 for more information on cluster node scoping.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: When using the WebInstaller, a value of true instructs the WebInstaller to place the Coherence library in the WEB-INF/lib directory of each Web application found in your J2EE application. If unspecified, this parameter defaults to false.</td>
</tr>
<tr>
<td>coherence-servletcontext-clustered</td>
<td>Either true or false to indicate whether the attributes of the ServletContext will be clustered. If true, then all serializable ServletContext attribute values will be shared among all cluster nodes. If unspecified, defaults to false, primarily because the Servlet specification indicates that the ServletContext attributes are local to a JVM and should not be clustered.</td>
</tr>
</tbody>
</table>

Table A–1 Configuration Parameters for Coherence*Web
Table A–1 (Cont.) Configuration Parameters for Coherence*Web

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>coherence-servletcontext-cachename (See Note 1)</td>
<td>The name of the Coherence cache to be used to hold the servlet context data if the servlet context is clustered.</td>
</tr>
<tr>
<td></td>
<td>If unspecified, defaults to servletcontext-storage. Appendix B, “Session Cache Configuration File” describes this parameter.</td>
</tr>
<tr>
<td>coherence-eventlisteners (See Note 1)</td>
<td>The comma-delimited list of names of application classes that want to receive events from the Web container. This list comes from the application listeners declared in the listener elements of web.xml.</td>
</tr>
<tr>
<td>coherence-enable-sessioncontext (See Note 1)</td>
<td>When set to true, allows the application to iterate sessions from the session context, thus disobeying the deprecation in the servlet specification.</td>
</tr>
<tr>
<td></td>
<td>If unspecified, defaults to false.</td>
</tr>
<tr>
<td>coherence-contextless-session-retain-millis</td>
<td>The number of milliseconds that a server holds a lock on a session while accessing that session without the session being implied by the current request context. A session is implied by the current request context if and only if the current thread is processing a Servlet request, and the request is associated with that session. All other access to a session object is &quot;out of context&quot;. For example, if a reference to an arbitrary session is obtained from a SessionContext object (if that option is enabled), or if the application has code that holds on to session object references to manage sessions directly. Since session access requires session ownership, &quot;out of context&quot; access to the session object automatically obtains ownership on behalf of the caller; that ownership will be retained for the number of milliseconds specified by this option so that repeated calls to the session do not individually obtain and release ownership, which is potentially an expensive operation. The legal range is 10 to 10000 (from 1/100th of a second up to 10 seconds).</td>
</tr>
<tr>
<td></td>
<td>If unspecified, defaults to 200.</td>
</tr>
<tr>
<td>coherence-session-cookies-enabled (See Note 1)</td>
<td>If unspecified, defaults to true to enable session cookies.</td>
</tr>
<tr>
<td>coherence-session-cookie-name (See Note 1)</td>
<td>The name of the session cookie.</td>
</tr>
<tr>
<td></td>
<td>If unspecified, defaults to JSESSIONID.</td>
</tr>
<tr>
<td>coherence-session-cookie-domain (See Note 1)</td>
<td>The domain of the session cookie as defined by RFC 2109. By default, no domain is set explicitly by the session management implementation.</td>
</tr>
<tr>
<td>coherence-session-cookie-path (See Note 1)</td>
<td>The path of the session cookie as defined by RFC 2109. By default, no path is set explicitly by the session management implementation.</td>
</tr>
<tr>
<td>coherence-session-cookie-max-age (See Note 1)</td>
<td>The maximum age in seconds of the session cookie as defined by RFC 2109. A value of -1 indicates that the cookie will not be persistent on the client; a positive value gives the maximum age that the cookie will be persisted by the client. Zero is not permitted.</td>
</tr>
<tr>
<td></td>
<td>If unspecified, defaults to -1.</td>
</tr>
<tr>
<td>coherence-session-urlencode-enabled (See Note 1)</td>
<td>When set to true, enables URL encoding of session ids.</td>
</tr>
<tr>
<td></td>
<td>If unspecified, defaults to true.</td>
</tr>
<tr>
<td>coherence-session-urlencode-name (See Note 1)</td>
<td>The parameter name to encode the session id into the URL with. On some containers, this value cannot be overridden.</td>
</tr>
<tr>
<td></td>
<td>If unspecified, defaults to jsessionid.</td>
</tr>
<tr>
<td>coherence-session-urldecode-bycontainer (See Note 1)</td>
<td>When set to true, uses the container’s decoding of the URL session ID. If coherence-session-urlencode-name has been overridden, this must be set to false. Setting this to false will not work in some containers.</td>
</tr>
<tr>
<td></td>
<td>If unspecified, defaults to true.</td>
</tr>
</tbody>
</table>
Coherence™Web Configuration Parameters

A-3

Table A–1 (Cont.) Configuration Parameters for Coherence™Web

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>coherence-session-urlencode-bycontainer</td>
<td>When set to true to use the container’s encoding of the URL session ID. Setting this to true may conflict with the setting for coherence-session-urlencode-name if it has been specified. If unspecified, defaults to false.</td>
</tr>
<tr>
<td>coherence-reaperdaemon-cluster-coordinated</td>
<td>When set to true, coordinates reaping in the cluster such that only one server will perform reaping within a given reaping cycle, and it will be responsible for checking all of the sessions that are being managed in the cluster. See “Cleaning Up Expired HTTP Sessions” on page 4-19 for more information on the session reaper. If unspecified, defaults to false.</td>
</tr>
<tr>
<td>coherence-reaperdaemon-sweep-modulo</td>
<td>This parameter is deprecated as of Coherence Release 3.5.</td>
</tr>
<tr>
<td>coherence-reaperdaemon-assume-locality</td>
<td>This setting allows the reaper to assume that the sessions that are stored on this node (for example, by a distributed cache service) are the only sessions that this node must check for expiry. This value must be set to false if the session storage cache is being managed by nodes that are not running a reaper, for example if cache servers are being used to manage the session storage cache. (It is suggested that if cache servers are being used, that the “split” model be selected, and that the session overflow storage be run in a separate distributed cache service that is managed entirely by the cache servers, while the session storage cache itself remain in a distributed cache service that is managed entirely by the application server JVMs to be able to take advantage of this “assume locality” feature.) See “Cleaning Up Expired HTTP Sessions” on page 4-19 for more information on the session reaper. If unspecified, defaults to true.</td>
</tr>
<tr>
<td>coherence-reaperdaemon-cycle-seconds</td>
<td>The number of seconds that the daemon rests between reaping. For production clusters with long session timeouts, this can safely be set higher. For testing, particularly with short session timeouts, it can be set much lower. Setting it too low can cause more network traffic and use more processing cycles, and only has benefit if the application requires the sessions to be invalidated quickly when they have expired. See “Cleaning Up Expired HTTP Sessions” on page 4-19 for more information on the session reaper. If unspecified, defaults to 300.</td>
</tr>
<tr>
<td>coherence-reaperdaemon-priority</td>
<td>The priority for the session reaper daemon. For more information, see the source for the java.lang.Thread class. If unspecified, defaults to 5.</td>
</tr>
<tr>
<td>coherence-session-cachename</td>
<td>This name overrides the name of the clustered cache that stores the sessions. If unspecified, defaults to session-storage. Appendix B, &quot;Session Cache Configuration File&quot; describes this parameter.</td>
</tr>
<tr>
<td>coherence-session-deathcert-cachename</td>
<td>This name overrides the name of the clustered cache that stores the IDs of &quot;recently departed&quot; sessions. If unspecified, defaults to session-death-certificates. Appendix B, &quot;Session Cache Configuration File&quot; describes this parameter.</td>
</tr>
<tr>
<td>coherence-session-management-cachename</td>
<td>This name overrides the name of the clustered cache that stores the management and configuration information for the session management implementation. Generally, it should be configured as a replicated cache. If unspecified, defaults to session-management. Appendix B, &quot;Session Cache Configuration File&quot; describes this parameter.</td>
</tr>
<tr>
<td>coherence-session-expire-seconds</td>
<td>This value overrides the session expiry time, and is expressed in seconds. Setting it to -1 causes sessions to never expire. If unspecified, defaults to 1800.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>coherence-session-id-length</td>
<td>This is the length, in characters, of generated session IDs. The suggested absolute minimum length is 8. If unspecified, defaults to 12.</td>
</tr>
<tr>
<td>coherence-shutdown-delay-seconds</td>
<td>This value determines how long the session management implementation waits before shutting down after receiving the last indication that the application has been stopped, either from ServletContextListener events (Servlet 2.3 or later) or by the destruction of Servlet and Filter objects. This value is expressed in seconds. A value of zero indicates synchronous shut-down; any positive value indicates asynchronous shut-down. If unspecified, defaults to 0, because some servers are not capable of asynchronous shut-down.</td>
</tr>
<tr>
<td>coherence-session-member-locking</td>
<td>This value, if set to true, prevents two threads in different JVMs from processing a request for the same session at the same time. A value of false is incompatible with sticky session optimizations and thread locking (that is, coherence-session-thread-locking and coherence-sticky-sessions should be set to false if this value is set to false). If unspecified, defaults to false.</td>
</tr>
<tr>
<td>coherence-session-app-locking</td>
<td>This value, if set to true, will prevent two threads in different applications from processing a request for the same session at the same time. A value of false is incompatible with thread locking. If set to true the value of the coherence-session-member-locking parameter will be ignored, as application locking implies member locking. If unspecified, defaults to false.</td>
</tr>
<tr>
<td>coherence-session-thread-locking</td>
<td>This value, if set to true, prevents two threads in the same JVM from processing a request for the same session at the same time. If set to true the value of the coherence-session-member-locking parameter is ignored, as thread locking implies member locking. If unspecified, defaults to true.</td>
</tr>
<tr>
<td>coherence-session-strict-spec</td>
<td>This value, if set to true, indicates that the implementation strictly adheres to the Servlet specification; setting it to false allows the implementation to ignore certain types of exceptions, instead of shutting down the application. If unspecified, defaults to true.</td>
</tr>
<tr>
<td>coherence-sticky-sessions</td>
<td>This value, if set to true, specifies whether sticky sessions optimizations will be used. This should only be enabled if a sticky load balancer is being used. If unspecified, defaults to false.</td>
</tr>
<tr>
<td>coherence-distributioncontroller-class</td>
<td>This value specifies a class name of the com.tangosol.coherence.servlet.HttpSessionCollection$SessionDistributionController interface implementation to use. This feature requires coherence-sticky-sessions optimization to be enabled. Legal values include: com.tangosol.coherence.servlet.AbstractHttpSessionCollection$DistributedController com.tangosol.coherence.servlet.AbstractHttpSessionCollection$HybridController com.tangosol.coherence.servlet.AbstractHttpSessionCollection$LocalController</td>
</tr>
</tbody>
</table>
Table A–1 (Cont.) Configuration Parameters for Coherence*Web

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| coherence-scopecontroller-class | This value specifies a class name of the optional com.tangosol.coherence.servlet.HttpSessionCollection$AttributeScopeController interface implementation to use.  
See “Session Attribute Scoping” on page 4-8 for more information.  
Legal values include:  
- com.tangosol.coherence.servlet.AbstractHttpSessionCollection$ApplicationScopeController  
- com.tangosol.coherence.servlet.AbstractHttpSessionCollection$GlobalScopeController |
| coherence-preserve-attributes   | This value, if set to true, specifies whether non-serializable attributes should be preserved as local ones. This feature requires coherence-sticky-sessions optimization to be enabled.  
If unspecified, defaults to false. |
| coherence-local-session-cachename | This name overrides the name of the local cache that stores non-distributed sessions when coherence-distributioncontroller-class parameter is specified.  
If unspecified, defaults to local-session-storage. Appendix B, “Session Cache Configuration File” describes this parameter. |
| coherence-local-attribute-cachename | This name overrides the name of the local cache that stores non-distributed sessions when either coherence-sessiondistributioncontroller-class parameter is specified or coherence-preserve-attributes parameter is true.  
If unspecified, defaults to local-attribute-storage. Appendix B, “Session Cache Configuration File” describes this parameter. |
| coherence-session-overflow-cachename | For the split model, this value overrides the name of the clustered cache that stores the “large attributes” that exceed a certain size and thus are determined to be more efficiently managed as separate cache entries and not as part of the serialized session object itself.  
If unspecified, defaults to session-overflow. Appendix B, “Session Cache Configuration File” describes this parameter. |
| coherence-attribute-overflow-threshold | For the split model, this value specifies the minimum length (in bytes) that the serialized form of an attribute value must be for that attribute value to be stored in the separate “overflow” cache that is reserved for large attributes.  
If unspecified, defaults to 1024. |

Notes:

1. This parameter does not control Coherence Web behavior when used with the WebLogic SPI implementation. If you are configuring Coherence Web using the WebLogic SPI implementation, see “Configuring Web Applications for Coherence*Web” on page 2-6 for more details on how to configure this attribute.
Session Cache Configuration File

Coherence*Web uses the caches and services defined in the `session-cache-config.xml` file to implement HTTP session management. This file is deployed under `WEB-INF/classes` in either the instrumented Web application or shared WebLogic Coherence*Web SPI library. Table B–1 describes the default cache-related values used in the `session-cache-config.xml` file.

### Table B–1 Cache-Related Values used in `session-cache-config.xml`

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session-management</td>
<td>This cache is used to store internal configuration and management information for the session management implementation. This information is updated infrequently; therefore, it is a replicated cache by default.</td>
</tr>
<tr>
<td>servletcontext-storage</td>
<td>If <code>ServletContext</code> attribute clustering (see the <code>coherence-servletcontext-clustered</code> parameter in Table A–1) is enabled (it is disabled by default), this cache is used to store <code>ServletContext</code> attributes. This cache is replicated by default, as it is expected that there will be a few read-mostly attributes.</td>
</tr>
<tr>
<td>session-storage</td>
<td>This cache is used to store session models. By default it is mapped to a near cache backed by a distributed cache since it is expected that a container will access and modify a subset of sessions multiple times (if sticky session load balancing is configured.) See “Session Models” on page 4-1 for more information on session models.</td>
</tr>
<tr>
<td>session-overflow</td>
<td>If the <code>coherence-sessioncollection-class</code> parameter (described in Table A–1) is set to <code>com.tangosol.coherence.servlet.SplitHttpSessionCollection</code>, then this cache will hold &quot;large&quot; session attributes. By default, session attributes larger than 1K will be stored in this cache. This is configured as a distributed cache.</td>
</tr>
<tr>
<td>session-death-certificates</td>
<td>Recently expired session IDs are stored in this cache to prevent reuse of a recently used session ID. By default, each storage node will hold up to 4000 session IDs, and session IDs will be evicted after 24 hours. This is configured as a distributed cache.</td>
</tr>
<tr>
<td>local-session-storage</td>
<td>This local cache is used to store session models that are considered to be &quot;local&quot; by the configured (if any) <code>coherence-distributioncontroller-class</code> parameter. Table A–1 describes this parameter.</td>
</tr>
<tr>
<td>local-attribute-storage</td>
<td>This local cache is used to store attributes that are not distributed. This can happen under two conditions:</td>
</tr>
<tr>
<td></td>
<td>■ A <code>coherence-distributioncontroller-class</code> is configured. Attributes for &quot;local&quot; sessions will be stored in this cache.</td>
</tr>
<tr>
<td></td>
<td>■ A non-serializable attribute is set on a distributed session. If <code>coherence-sticky-sessions</code> and <code>coherence-preserve-attributes</code> are set to <code>true</code>, then this attribute will be placed in this cache. These parameters are described in Table A–1.</td>
</tr>
</tbody>
</table>
Table B–2 describes the services-related values used in the session-cache-config.xml file.

### Table B–2 Services-Related Values used in session-cache-config.xml

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReplicatedSessionsMisc</td>
<td>This replicated service is used by the session-management and servletcontext-storage caches.</td>
</tr>
<tr>
<td>DistributedSessions</td>
<td>This distributed service is used by the following caches:</td>
</tr>
<tr>
<td></td>
<td>• session-storage</td>
</tr>
<tr>
<td></td>
<td>• session-overflow</td>
</tr>
<tr>
<td></td>
<td>• session-death-certificates</td>
</tr>
<tr>
<td></td>
<td>The tangosol.coherence.session.localstorage system property controls whether a JVM stores and manages data for these caches. Under most circumstances, this should be set to false for Web container JVMs. See &quot;Deployment Topologies&quot; on page 4-14 for more details.</td>
</tr>
<tr>
<td>SessionOwnership</td>
<td>This invocation service is used by the sticky session optimization feature (if coherence-sticky-sessions is set to true).</td>
</tr>
</tbody>
</table>

Example B–1 illustrates the contents of the session-cache-config.xml file. The cache- and services-related values described in the previous tables appear in **bold**.

**Example B–1 Contents of the session-cache-config.xml File**

```xml
<?xml version="1.0"?>
<!DOCTYPE cache-config SYSTEM "cache-config.dtd">
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<!--                                                                       -->
<!--      Cache configuration descriptor for Coherence*Web                 -->
<!--                                                                       -->
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
<cache-config>
  <caching-scheme-mapping>
    <!-- The clustered cache used to store Session management data. -->
    <cache-mapping>
      <cache-name>session-management</cache-name>
      <scheme-name>replicated</scheme-name>
    </cache-mapping>

    <!-- The clustered cache used to store ServletContext attributes. -->
    <cache-mapping>
      <cache-name>servletcontext-storage</cache-name>
      <scheme-name>replicated</scheme-name>
    </cache-mapping>

    <!-- The clustered cache used to store Session attributes. -->
    <cache-mapping>
      <cache-name>session-storage</cache-name>
      <scheme-name>session-near</scheme-name>
    </cache-mapping>
  </caching-scheme-mapping>
</cache-config>
```
The clustered cache used to store the "overflowing" (split-out due to size) Session attributes. Only used for the "Split" model.

```xml
<cache-mapping>
  <cache-name>session-overflow</cache-name>
  <scheme-name>session-distributed</scheme-name>
</cache-mapping>
```

The clustered cache used to store IDs of "recently departed" Sessions.

```xml
<cache-mapping>
  <cache-name>session-death-certificates</cache-name>
  <scheme-name>session-certificate</scheme-name>
</cache-mapping>
```

The local cache used to store Sessions that are not yet distributed (if there is a distribution controller).

```xml
<cache-mapping>
  <cache-name>local-session-storage</cache-name>
  <scheme-name>unlimited-local</scheme-name>
</cache-mapping>
```

The local cache used to store Session attributes that are not distributed (if there is a distribution controller or attributes are allowed to become local when serialization fails).

```xml
<cache-mapping>
  <cache-name>local-attribute-storage</cache-name>
  <scheme-name>unlimited-local</scheme-name>
</cache-mapping>
```

Replicated caching scheme used by the Session management and ServletContext attribute caches.

```xml
<replicated-scheme>
  <scheme-name>replicated</scheme-name>
  <service-name>ReplicatedSessionsMisc</service-name>
  <backing-map-scheme>
    <local-scheme>
      <scheme-ref>unlimited-local</scheme-ref>
    </local-scheme>
  </backing-map-scheme>
  <request-timeout>30s</request-timeout>
  <autostart>true</autostart>
</replicated-scheme>
```

Near caching scheme used by the Session attribute cache. The front cache uses a Local caching scheme and the back cache uses a Distributed caching scheme.

```xml
<near-scheme>
```

Session Cache Configuration File  B-3
<scheme-name>session-near</scheme-name>
<front-scheme>
<local-scheme>
<scheme-ref>session-front</scheme-ref>
</local-scheme>
</front-scheme>
<back-scheme>
<distributed-scheme>
<scheme-ref>session-distributed</scheme-ref>
</distributed-scheme>
</back-scheme>
<invalidation-strategy>present</invalidation-strategy>
</near-scheme>

<local-scheme>
<scheme-name>session-front</scheme-name>
<eviction-policy>HYBRID</eviction-policy>
<high-units>1000</high-units>
<low-units>750</low-units>
</local-scheme>

<distributed-scheme>
<scheme-name>session-distributed</scheme-name>
<scheme-ref>session-base</scheme-ref>
<backing-map-scheme>
<local-scheme>
<scheme-ref>unlimited-local</scheme-ref>
</local-scheme>
<!-- for disk overflow use this backing scheme instead: -->
<overflow-scheme>
<scheme-ref>session-paging</scheme-ref>
</overflow-scheme>
</backing-map-scheme>
</distributed-scheme>

<!-- Distributed caching scheme used by the "recently departed" Session cache. -->
<distributed-scheme>
<scheme-name>session-certificate</scheme-name>
<scheme-ref>session-base</scheme-ref>
<backing-map-scheme>
<local-scheme>
<eviction-policy>HYBRID</eviction-policy>
<high-units>4000</high-units>
<low-units>3000</low-units>
<expiry-delay>86400</expiry-delay>
</local-scheme>
</backing-map-scheme>
</distributed-scheme>

<!-- "Base" Distributed caching scheme that defines common configuration. -->
<distributed-scheme>
<scheme-name>session-base</scheme-name>
<service-name>DistributedSessions</service-name>
<thread-count>0</thread-count>
<lease-granularity>member</lease-granularity>

B-4 Oracle Coherence User’s Guide for Oracle Coherence*Web
<local-storage system-property="tangosol.coherence.session.localstorage">false</local-storage>
<partition-count>257</partition-count>
<backup-count>1</backup-count>
<backup-storage>
  <type>on-heap</type>
</backup-storage>
<backing-map-scheme>
  <local-scheme>
    <scheme-ref>unlimited-local</scheme-ref>
  </local-scheme>
</backing-map-scheme>
<request-timeout>30s</request-timeout>
<autostart>true</autostart>
</distributed-scheme>

<!--
Disk-based Session attribute overflow caching scheme.  -->
<overflow-scheme>
  <scheme-name>session-paging</scheme-name>
  <front-scheme>
    <local-scheme>
      <scheme-ref>session-front</scheme-ref>
    </local-scheme>
  </front-scheme>
  <back-scheme>
    <external-scheme>
      <bdb-store-manager/>
    </external-scheme>
  </back-scheme>
</overflow-scheme>

<!--
Local caching scheme definition used by all caches that do not require an eviction policy. -->
<local-scheme>
  <scheme-name>unlimited-local</scheme-name>
  <service-name>LocalSessionCache</service-name>
</local-scheme>

<!--
Clustered invocation service that manages sticky session ownership.  -->
<invocation-scheme>
  <service-name>SessionOwnership</service-name>
  <request-timeout>30s</request-timeout>
</invocation-scheme>
Index

A
Ant task, WebInstaller, 3-5
Apache Tomcat, 1-2
application server-scoped cluster nodes, 4-9
  packing and configuring, 2-5
ApplicationScopeController interface, 2-7, A-5
AttributeScopeController interface, 4-8

C
cache server, starting, 2-4
Caucho Resin, 1-2
ClassLoader interface, 4-6
cluster node isolation, 4-8
Coherence*Web
  defined, 1-1
Coherence*Web configuration parameters, A-1
Coherence*Web SPI, 2-1
  configuration and deployment overview, 2-2
  configuration for WebLogic Server 10.3 and 9.2
    MPI, 2-1
  limitations, 2-10
  location, 2-1
  overview, 2-1
  packing and configuration, 2-5
  requirements, 2-1
  WebLogic Server-specific options, 2-6
coherence-attribute-overflow-threshold parameter, A-5
coherence-cluster-owned parameter, A-1
coherence-contextless-session-retain-millis parameter, A-2
coherence-distributioncontroller-class parameter, A-4
coherence-enable-sessioncontextparameter, 2-8, A-2
coherence-eventlisteners parameter, 2-8, A-2
coherence-factory-class parameter, A-1
coherence-jar, 2-1, 2-5, 2-6, 2-9, 4-9, 4-10, 4-11, 5-1
coherence-local-attribute-cachename parameter, A-5
coherence-local-session-cachename parameter, A-5
coherence-preserve-attributes parameter, 2-7, A-5
coherence-reaperadaemon-assume-locality option, 4-20, 4-22
coherence-reaperadaemon-assume-locality parameter, 2-7, A-3
coherence-reaperadaemon-cluster-coordinated option, 4-20
coherence-reaperadaemon-cluster-coordinated parameter, A-3
coherence-reaperadaemon-cycle-seconds option, 4-22
coherence-reaperadaemon-cycle-seconds parameter, A-3
coherence-reaperadaemon-cycle-seconds property, 4-20
coherence-reaperadaemon-priority parameter, A-3
coherence-reaperadaemon-sweep-modulo parameter, A-3
coherence-scopecontroller-class parameter, 2-7, A-5
coherence-servletcontext-cachename parameter, 2-8, A-2
coherence-servletcontext-clustered parameter, 2-8, A-1
coherence-session-app-locking parameter, 2-7, A-4
coherence-session-cachename parameter, A-3
coherence-sessioncollection-class parameter, 2-7, A-1
coherence-session-cookie-domain parameter, 2-8, 4-6, 4-8, A-2
coherence-session-cookie-max-age parameter, 2-8, A-2
coherence-session-cookie-name parameter, 3-9, A-2
coherence-session-cookie-path parameter, 2-8, 4-6, 4-7, A-2
coherence-session-cookies-enabled parameter, 2-8, 3-9, A-2
coherence-session-deathcert-cachename parameter, A-3
coherence-session-expire-seconds option, 4-19
coherence-session-expire-seconds parameter, A-3
coherence-session-id-length parameter, 2-9, A-4
coherence-session-management-cachename parameter, A-3
coherence-session-member-locking parameter, 2-7, 4-12, 4-13, A-4
coherence-session-overflow-cachename parameter, A-5
coherence-session-strict-spec parameter, A-4
coherence-session-thread-locking parameter, 4-13, A-4
coherence-session-urldecode-bycontainer parameter, 2-9, A-2
coherence-session-urlencode-bycontainer
parameter, 2-9, A-3
coherence-session-urlencode-enabled
parameter, 2-8, A-2
coherence-session-urlencode-name parameter, 2-9, A-2
coherence-shutdown-delay-seconds parameter, A-4
cohereance-sticky-sessions parameter, 2-7, 4-14, A-4
cohereance-web.jar, 2-9, 3-2
cohereance-web-sessions-enabled parameter, 5-2
cohereance-web-spi.jar, 2-9
cohereance-web-spi.war, 2-1, 2-2, 2-5, 2-6, 2-9, 5-2
cohereance-web.xml, 3-5, 3-8
cohereance-wlp.jar, 5-1
CollectionClassName MBean attribute, 4-17
ConcurrentModificationException class, 4-12
configuration parameters, A-1
collection-param element, 2-6

D
DefaultFactory interface, A-1
deployment topologies, 4-14
descriptor attribute, 3-5
DistributedController interface, A-4
DistributedSessions value, B-2

E
EAR-scoped cluster nodes, 4-10
packing and configuring, 2-6

F
FactoryClassName MBean attribute, 4-17

G
GlobalScopeController interface, 4-8, A-5

H
HTTP session management, testing, 3-7
HttpSession interface, 2-4
HttpSessionAttributeListener class, 3-8
HttpSessionCollection interface, 4-1, A-1
HttpSessionListener class, 3-8
HttpSessionManagerMBean interface, 4-16
HttpSessionManagerMBean, attributes and operations, 4-16
HttpSessionModel interface, 4-1
HybridController interface, A-4

I
in-process deployment topology, 4-14
installation
Coherence*Web SPI, 2-1
WebInstaller, 3-1
Cauchos Resin 3.1.x, 3-2
general instructions, 3-3

J
JMX management framework, 4-16

L
LastAccessedTime property, 4-19
lease-granularity element, 4-12, 4-13
load balancer, 3-7
command line options, 3-8
LocalAttributeCacheName MBean attribute, 4-17
LocalAttributeCount MBean attribute, 4-17
LocalController interface, A-4
LocalSessionCacheName MBean attribute, 4-17
LocalSessionCount MBean attribute, 4-17
local-session-storage value, B-1

M
MaxInactiveInterval property, 4-19
MBeans, 4-16
member session locking mode, 4-12
monolithic session model, 4-3
MonolithicHttpSessionCollection interface, 4-3, A-1
MonolithicHttpSessionModel interface, 4-3

O
operations attribute, 3-5
optimistic session locking mode, 4-12
out of process deployment topology, 4-14
out-of-process with Coherence*Extend deployment topology, 4-15
OverflowAverageSize MBean attribute, 4-17
OverflowCacheName MBean attribute, 4-17
OverflowMaxSize MBean attribute, 4-17
OverflowThreshold MBean attribute, 4-17
OverflowUpdates MBean attribute, 4-17

P
p13n-cache-config.xml file, 5-1
PartitionedIterator class, 4-21

R
ReplicatedSessionsMisc value, B-2
resetStatistics MBean operation, 4-18

S
SAML Single Sign-On (SSO), 2-9
saml2.war, 2-9
server type alias, 1-2
ServletContext ("global") attributes, 3-4
ServletContextAttributeListener class, 3-8
ServletContextCacheName MBean attribute, 4-18
ServletContextListener class, 3-8
ServletContextListener events, A-4
ServletContextName MBean attribute, 4-18
servletcontext-storage value, B-1
ServletRequestAttributeListener class, 3-8
ServletRequestListener class, 3-8
session attribute scoping, 4-6, 4-8
session cookies, 4-7
session locking modes, 4-12
session models, 4-1
session reaper, 4-20
configuring, 4-22
session scoping, 4-6
SessionAverageLifetime MBean attribute, 4-18
SessionAverageSize MBean attribute, 4-18
session-cache-config.xml file, 2-2, 2-4, 2-9, 4-6
session-cache-config.xml parameters, B-1
SessionCacheName MBean attribute, 4-18
session-death-certificates value, B-1
session-descriptor element, 2-7
SessionDistributionController interface, A-4
SessionHelper.Factory interface, A-1
SessionIdLength MBean attribute, 4-18
session-management value, B-1
SessionMaxSize MBean attribute, 4-18
SessionMinSize MBean attribute, 4-18
session-overflow value, B-1
SessionOwnership value, 4-14, B-2
SessionServlet class, 3-9
SessionStickyCount MBean attribute, 4-18
session-storage value, B-1
SessionTimeout MBean attribute, 4-18
SessionUpdates MBean attribute, 4-18
Single Sign-On (SSO), 2-9
Single Sign-On for WebLogic Server, 2-10
Smart Update, 2-2
split session model, 4-4
SplitHttpSessionCollection interface, 2-7, 4-4, 4-17, A-1
SplitHttpSessionModel interface, 4-4
starting a cache server, 2-4
sticky sessions, 4-13, A-4
Sun Microsystems JVM, 2-4

T
testing HTTP session management, 3-7
thread session locking mode, 4-13
timeout system property, 4-13
topologies, deployment, 4-14
traditional session model, 4-2
TraditionalHttpSessionCollection interface, 4-2, A-1
TraditionalHttpSessionModel interface, 4-2

W
WAR-scoped cluster nodes, 4-11
    packing and configuring, 2-6